The Resource Journal For IBM PC Programmers

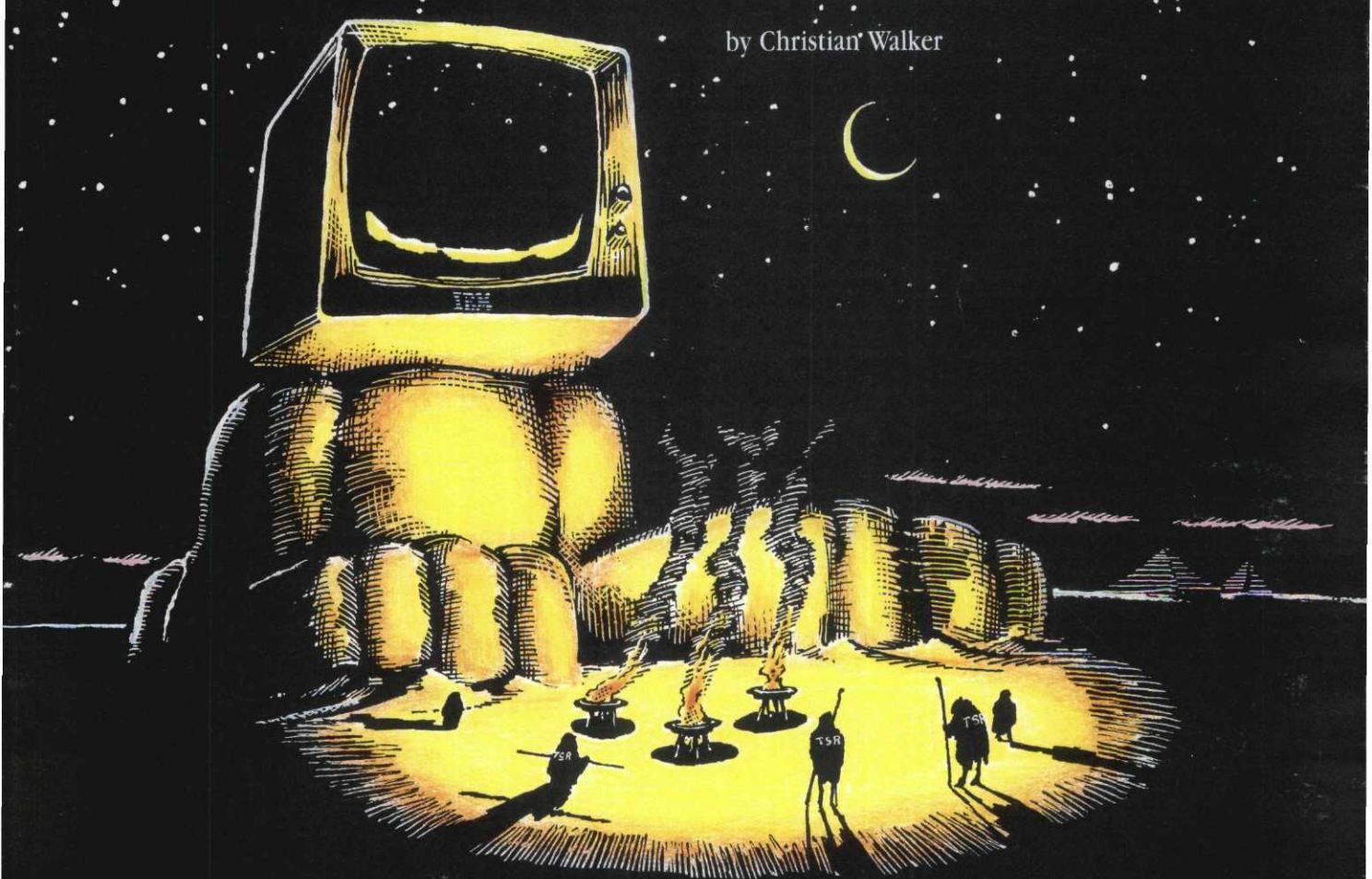
PROGRAMMER'S Journal

EPTEMBER/OCTOBER 1987
VOLUME 5.5
$3.95

Slouching towards the 386

The 80386 and OS/2

by Christian Walker



TSRs, Expanded Memory, and Turbo Pascal

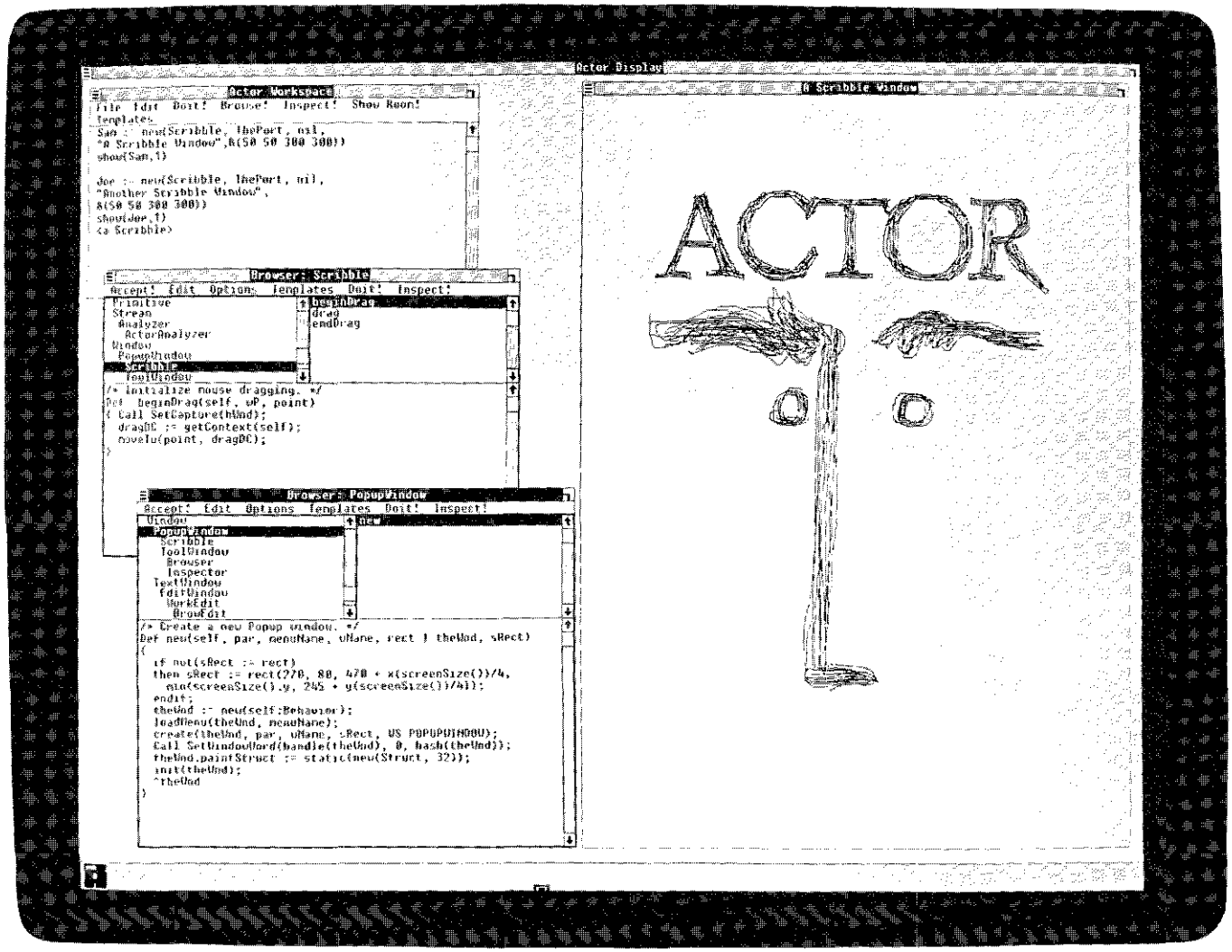
by John L. Newlin

The Case of the Invisible CapsLock

by Tom Swan

PRESERVED BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOW TO WRITE A WINDOWS APPLICATION IN TEN MINUTES.



Actor™ is a new language that combines Microsoft® Windows with object-oriented programming. This means you can produce mouse and window applications very quickly.

For example, we created a simple "paint" program, and used it to draw the Actor logo you see on the screen. The whole program only took ten lines and ten minutes. Part of it is in the middle window on the left.

Above, you see the commands that initialized the paint window and made it appear on the screen. Below, some code that's built into Actor, specifying window behavior. Through a process known as "inheritance," it's called into play automatically.

Try programming in this new way, and you'll never go back.

Find out about Actor. Call The Whitewater Group®, (312) 491-2370.

Technology Innovation Center
906 University Place, Evanston, IL 60201

MICROSOFT® WINDOWS

Number **73** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EVERYTHING THE C PROGRAMMER NEEDS

Eco-C88 C Compiler:

A full, professional C compiler with many ANSI enhancements at an unbelievably low price.

* Prototyping, enum, void data types, plus structure passing & assignment * All operators and data types * Over 200 library functions * cc and mini-make for easy use * 8087 support * ASM or OBJ output * Lint-like tiered error messages * Fast code * CED editor (edit-compile-link from within the editor) * Expanded user's manual * Not copy protected * All for only $59.95!

Eco-C88 Flexi-Graph Graphics Package

Everything you need to write dramatic graphics effects into your Eco-C88 C programs.

* EGA, CGA, and Z100 support * Over 100 graphics functions (many are PLOT-10 compatible) * Most assembler support routines are outside small model code-data * Write thru BIOS (for compatibility) or to memory (for speed) * Graphics function help from CED editor * World, pixel or turtle color graphics modes * 47 standard fill patterns, 17 line dashing patterns, Hershey fonts, plus user defineable fill, dash, and fonts * Supports view areas, rotateable fonts, clipping, arbitrary fill areas, extensive error checking, examples, and user's manual * Only $39.95

Eco-C88 Windowing Library

Use this library to build pop-up windows, help windows, selection menus, special effects—anywhere you need an attention getter.

* CGA and EGA support * Control any program that goes through the BIOS * Use up to 255 windows * No special window commands — use plain old printf( ) to write to a window * Resize and move windows * Custom window titles and borders * Can be used with ANSI device driver * Most window code-data are outside small model * User's manual and examples for only $29.95

Ecosoft Librarian

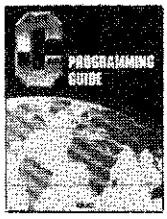
Combine your modules, functions, and subroutines into your own library for easy link commands. Compatible with any standard MSDOS OBJ files * Add, delete, and extract from a library * Get table of contents or index of a library * Combine libraries, control library page size, use switches for combinations, process complex library requests, use wildcards, and do library directives from command files * Complete with user's manual for only $29.95

Developer's Library

Contains the source code for for all library functions, including the transcedentals, memfiles, and those written in assembler. $25.00 with order, $50.00 if ordered later. (Sold only to Eco-C88 owners.)

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

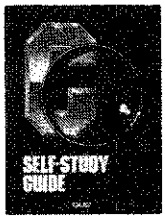
o-C88 C compiler requires an IBM PC, XT, or AT (or compatible) with 5K of memory, 2 disk drives and MSDOS 2.1 or later.



C Programming Guide

(Purdum, Que Corp.)

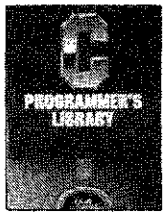
The second edition of the B. Dalton bestseller. Perfect for those just getting started with C. Includes discussion of many X3J11 ANSI Standards Committee recommendations. Many error messages from Eco-C88 give page references to this book. Price is $20.00 plus $2.00 shipping.



C Self-Study Guide

(Purdum, Que Corp.)

Using a question-answer approach, this book is filled with shortcuts, tips, techniques, and traps to avoid when learning C. Price is $17.00 plus $2.00 for shipping.



C Programmer's Library

(Purdum, Leslie, Stegemoller, Que Corp.)

Another B. Dalton bestseller. An intermediate C text for the programmer that wants to get the most from the language. Contains source code for many functions including an ISAM file handler. Price is $22.00 plus $2.00 for shipping.

To order, call or write:

1-800-952-0472 (for orders) or **1-317-255-6476** (tech. info.)

ORDER FORM CLIP & MAIL TO:
Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46220

- ☐ Eco-C88 C Compiler $59.95 _____
- ☐ Eco-C88 Flexi-Graph Graphics Package $39.95 _____
- ☐ Eco-C88 Windowing Library $29.95 _____
- ☐ Ecosoft Librarian $29.95 _____
- ☐ Developer's Library $25.00 ($50.00 if not with order) _____
- ☐ C Programming Guide $20.00 _____
- ☐ C Self-Study Guide $17.00 _____
- ☐ C Programmer's Library $22.00 _____

SHIPPING $4.00

TOTAL (IND. RES. ADD 5% TAX) _____

PAYMENT: ☐ VISA ☐ MC ☐ AE ☐ CHECK

CARD# _____ EXPIR. DATE _____

NAME _____

ADDRESS _____

CITY _____ STATE _____

ZIP _____ PHONE _____

Number 8 on the Reader Service Card

ECOSOFT

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal For IBM PC Programmers

PROGRAMMER'S Journal

Volume 5.5
September/October 1987

Contents

4 **Note from the Editors**

6 **Letters**

10 **PJ Forum:** Slouching towards the 386—The Intel 80386 Processor versus IBM's Operating System 2 *by Christian Walker*
History awaits IBM's Second Coming

16 **PJ Interview:** An Interview with Jack Davis, President of Metagraphics, Or, I Never Meta Man I Didn't Like *by Frank D. Greco*
Frank opens a window on an innovative graphics company

20 **Consulting:** Making Computer Tools Pay for Themselves *by Paul Barkley*
When does it pay to change tools?

24 **ASM:** The Case of the Invisible CapsLock *by Tom Swan*
A TSR to keep your eyes on the screen

28 **Pascal:** TSRs Using Expanded Memory with Turbo Pascal *by John J. Newlin*
Further TSR explorations, off into space

42 **Pascal:** A Note on Optimizing Turbo Pascal *by Jon Greenblatt*
Useful suggestions from a *PJ* reader

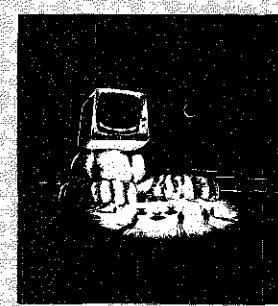
44 **C:** The Proposed ANSI C Language Standard *by Rex Jaeschke*
Rex reveals **limits.h** and **float.h** headers

48 **C:** A Note on Printer Output from Turbo C Programs *by Bernard H. Robinson, Jr.*
Printer help for the new Turbo C user

52 **BASIC:** A Matchup of Turbo Basic 1.0 and QuickBASIC 3.0 *by Paul P. Dymerski*
A BASIC showdown—who will finish first?

60 **Miscellaneous** *by M. Steven Baker*
Our technical editor identifies items of interest

64 **PJ Billboard**



At the foot of the IBM sphinx, a small tribe of TSRs has lived in peace and harmony for countless eons. In the night, the Sphinx suddenly stood up and stretched, then lay down again. The TSRs look on with fear and puzzlement. None of them has been injured as yet, but what will the future hold? The Sphinx, as usual, remains silent. *Cover by Jim Carpenter.*

Page 10

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Interlocking Pieces:

Blaise and Turbo Pascal.

Whether you're a Turbo Pascal expert or a novice, you can benefit from using special tools to enhance your programs, make them reliable, and give them a professional look. With windows, menus, pop-up memory resident programs, and communications support, Blaise Computing offers you all the right pieces to solve your Turbo Pascal development puzzle.

◆POWER TOOLS PLUS™ is a general purpose library of over 140 functions and procedures, like fast direct video access, general screen handling including multiple monitors, and EGA 43-line text mode. Stackable and removable windows with optional borders and cursor memory provide complete windowing capabilities. Pop-up menus are easily incorporated into your programs using the menu management functions.

POWER TOOLS PLUS allows you to inspect, allocate, and free DOS memory. Execute any program, including internal DOS commands, from within your Turbo Pascal program.

Write general interrupt service routines in Turbo Pascal for either hardware or software interrupts using the POWER TOOLS PLUS ISR support functions. Blaise Computing's unique intervention code lets you develop memory resident applications that take full advantage of DOS capabilities. With simple procedure calls, "schedule" a Turbo Pascal procedure to execute either when a "hot key" is pressed or at a specified time.

◆ASYNCH PLUS™ provides the crucial core of hardware interrupts needed to support asynchronous data communication. This package offers simultaneous buffered input and output to both COM ports at speeds up to 9600 baud, and supports the XON/XOFF protocol, XMODEM file transfer and modem control functions.

The underlying functions of ASYNCH PLUS are carefully crafted in assembler and drive the UART and programmable interrupt controller chips directly. These functions, installed as a memory resident system, require just 3.2K bytes. Complete Turbo Pascal interfaces to the assembler routines are provided.

The source code is included in all Blaise Computing products and is efficiently crafted, readable and easy to modify. The INCLUDE utility is provided to automatically determine which procedures and functions are required, and to compact them all into a single file to be included in your application. Each package is accompanied by an indexed manual describing each procedure and function in detail with example code fragments. The documentation reflects the attention to detail and commitment to technical support that have distinguished Blaise Computing over the years.

Turbo ASYNCH PLUS and Turbo POWER TOOLS PLUS support Turbo Pascal Version 2.00 and later and require DOS 2.00 or later.

Turbo ASYNCH and Turbo POWER TOOLS PLUS are just $99.95 each. Buy them together for only $175.00. Or for only $199.00, receive PLUSPAK™ a comprehensive package of ASYNCH PLUS, POWER TOOLS PLUS and KeyPlayer — *the right* pieces to put your Turbo Pascal puzzle together.

BLAISE MENU

Turbo SAVER SCREEN
COMING SOON! General screen management, paint screens, block mode data entry or field-by-field control with instant screen access. For Turbo Pascal.

EXEC *$95.00*
NEW VERSION! Program chaining executive. Chain one program from another in different languages, specify common data areas, less than 2K of overhead.

KeyPlayer *$49.95*
"Super-batch" program. Create batch files which can invoke programs and provide input to them, run any program unattended, create demonstration programs, analyze keyboard usage.

RUNOFF *$49.95*
Text formatter for all programmers. Written in Turbo Pascal; flexible printer control, user-defined variables, index generation, and a general macro facility.

Turbo C TOOLS *$129.00*
Windows; ISRs; intervention code; screen handling and EGA 43-line text mode support; direct screen access; DOS file handling and more. For Turbo C.

PASCAL TOOLS/TOOLS 2 *$175.00*
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS Pascal.

C TOOLS PLUS *$175.00*
Windows; ISRs; screen handling; multiple monitors; EGA 43-line text mode; direct screen access; DOS file handling and more. For MS and Lattice C version 3.00 and later.

VIEW MANAGER *$275.00*
General screen control; paint screens; block mode data entry or field-by-field control with instant screen access. For C or MS Pascal.

ASYNCH MANAGER *$175.00*
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF, up to 9600 baud; modem control and XMODEM. For C or MS Pascal.

FLIGHT TOOLS *$99.95*
Windows; ISRs; EGA 43-line text mode; direct screen access; DOS file handling and more. For the Datalight C compiler.

TO ORDER CALL TOLL FREE
800-333-8087
TELEX NUMBER 334580

YES! Send me the right pieces!
Enclosed is $_____ for _____ copies of _____.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
Name: _____ Phone: (_____) _____
Address: _____
City: _____ State: _____ Zip: _____
VISA or MC#: _____ Exp. Date: _____

Turbo Pascal is a trademark of Borland International.

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441
Number **2** on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal For IBM PC Programmers
PROGRAMMER'S Journal

Editors
Charles Aylworth, Erin O'Connor
Technical Editor
M. Steven Baker
Contributing Editors
Frank D. Greco, Tom Swan

Contributing Writers

Paul Barkley
Paul P. Dymerski
Frank D. Greco
Jon Greenblatt
Rex Jaeschke
John J. Newlin
Bernard H. Robinson, Jr.
Tom Swan
Christian Walker

Publisher
Elizabeth Oakley
Production
Ralph Bentley
Illustration
Jim Carpenter
Advertising
Robert Tappan
Subscriptions
Gary Hoyer
Circulation
Preston Moser
Founder
Greg Estes

Business and Editorial Offices:
Programmer's Journal
150 North 4th Street
Springfield, OR 97477
(503) 747-0800

Change of Address:
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403
(503) 747-0800

Subscriptions:
1 year (6 issues) $24 U.S. ($30 in Canada and $40 elsewhere)
2 years (12 issues) $44 U.S. ($55 in Canada and $75 elsewhere)
3 years (18 issues) $64 U.S. ($80 in Canada and $110 elsewhere)

Articles and letters in Programmer's Journal represent the views of their authors and not necessarily those of the publisher. Letters to the editor become the property of Programmer's Journal and may be edited and published.
Advertising: For ad rates, deadlines, and placement, call Robert Tappan, Ralph Bentley, or Gary Hoyer (503) 747-0800, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.
IBM and Personal Computer are trademarks of International Business Machines, Inc.; dBASE II and dBASE III are trademarks of Ashton-Tate.
Programmer's Journal (ISSN 0747-5861) is published bi-monthly by Oakley Publishing Company. Second class postage paid at Eugene, OR, and additional mailing offices.
POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.
©1987 OAKLEY PUBLISHING COMPANY

NOTE FROM THE EDITORS

Hi. My name is Charles Aylworth. This is my first issue as editor of *PJ*, and I'd like to talk about where we're going. Erin O'Connor and I are sharing the enjoyable task of bringing *PJ* to you every other month. We are starting off with, in my opinion, a great magazine. We owe a word of thanks to Robert Keller, who has guided *PJ* through the last couple of years with real success. Robert is a tough act to follow and has set us a high mark to achieve.

There are changes right and left around *PJ*. But first, let's be sure about what isn't going to change. We intend to bring you the magazine you have come to know, and we hope, to love. *PJ* isn't going to change its style or the quality articles you expect. Everything else has changed, or can be expected to change, in the near future. We have a new address (we needed more space), a new phone number (we moved across the river), some new departments (more later), and some new ideas. Soon we will be moving into the computer age, putting the magazine together with a desktop publisher.

Some familiar names are missing from the contents page this issue. Fear not, we aren't dropping our well known authors. Each of the missing brotherhood has his own sad story. Changing jobs, launching products, just too darn busy, started writing for this new magazine and can't make the deadline, every excuse in the book. The good news is, almost no one is mad at *PJ*, and all will be back in the fold in the near future.

At the same time, we are welcoming some new authors. Christian Walker leads off with a history of the Intel line of microprocessors and some musings on OS/2. You can look forward to hearing more from Chris in future issues. Our technical editor Steven Baker will be writing the MISCELLANEOUS department on a regular basis. We are hearing from new authors all the time, and we hope it never stops.

As we all know, IBM has launched itself into a new era. Our motto is "The Resource Journal for IBM PC Programmers." Should *PJ* concentrate on the PC, or jump on the PS/2-OS/2 bandwagon? Well, we are definitely not going to abandon the PC, IBM notwithstanding. There's still a lot of nifty code waiting to be written for today's machine. At the same time, we can't ignore the new breed, and as Chris Walker's article demonstrates, we don't intend to. But what's the right mix?

In this and future issues, you will see lots of little changes. A line here, a department there—we're going to keep *PJ* active and growing. The key to all of this is you, the reader. We need to know what you think, and what you want. We appreciate your taking the time to fill out the Peer Review card. We're going to try to make it easier for you.

The key word for *PJ* is feedback. The more we hear from you, the more the magazine will reflect what you want. If you have an idea, a gripe, or an article, don't be shy, send it in. For now then, see ya'later. I'm glad to be aboard.

—Chuck

I'd like to add a few words—of gratitude to Robert Keller under whom it was an education and a pleasure to work, and of welcome to Chuck Aylworth, who brings enthusiasm, a discerning eye, and a wealth of experience to *PJ*.

This seems to be the year of the rough beast (no reference to Robert's and Chuck's beards). Robert Keller thought the "new DOS" looked a little hairy and added the blurb "the protected mode slouches towards your CPU" to Frank Greco's article on DOS for the 286 in the January/February 1987, issue of *PJ*. Then Hal Hardenbergh in a fit of (unwarranted and uncharacteristic) self-deprecation and "with sincere apologies to William Butler Yeats" led off the manual for his new DBASIC for the Atari ST with the epigraph ". . .what rough BASIC, its hour come round at last, slouches CPUwards to be born?" Now Chris Walker has sent us a fine article on the Intel processors and OS/2 that he suggests we title "Slouching towards the 386." We still think it's a good line, and appropriate, so we'll add our apologies to Yeats and make amends in this issue by printing the poem we've all so barbarously paraphrased. Other lines in that rhetorical masterpiece will ring some bells, too.

Bill Hunt is no slouch (ouch!). We've reprinted many articles from the *Lattice C Users' Group* newsletter in our C department, and you must have noticed that the *LCUG* newsletter did not confine itself to articles on Lattice C. The de facto *Lattice and Microsoft C Users' Group* newsletter now appears under that name officially. We recommend the *LMCUG* newsletter unreservedly. To subscribe, write to William J. Hunt, Lattice and Microsoft C Users' Group, P.O. Box 271965, Concord, CA 94527.

—Erin □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Software Family

Dept PJ97, 649 Mission Street
San Francisco, CA 94105

(1)800-443-7176

In California or outside U.S.
(415)583-4166

Trust TSF to provide the best value and service!

* Technical advice and support by programmers
* Honest and equitable business practices
* Prompt, flexible service to meet your needs

We accept checks, Visa, MasterCard, American Express and COD. We charge your card only when we ship and do **not** add a surcharge. We provide free UPS delivery on software orders over $100 ($3 delivery charge on orders under $100). We add actual charges for UPS Blue Label or Federal Express rush service. Our COD fee is $2. We allow return privileges on most products. We carry a large inventory and ship promptly, so you receive your shipment within 3 to 6 working days of placing your order. Give us a try!

Turbo C Specials

Borland **Turbo C** (list $99) **$59**

Blaise **Turbo C Tools** (list $129) **$95**
ISR/TSR support, critical error trapping, direct video access, more

Creative **Vitamin C** (list $225) **$150**
Windows, menus and data entry with source

Gimpel **PC Lint** (list $139) **$95**
Checks all modules of a C program to warn if function calls do not include the proper number and types of parameters, if global variables are declared inconsistently or if you have used common C "gotchas"

Metagraphics **Turbo Window/C** ($95) **$75**
Graphics windows, menus and multiple fonts

Softfocus **Btree & ISAM** (list $115) **$99**
Complete multi-key ISAM file system with source

Prices good until October 15, 1987

Basic Turbo Pascal Publishing C dBase AI

Aldebaran **Source Print** (list $75) **$59**
Aldebaran **Tree Diagrammer** (list $55) **$42**
Blaise **C Tools +** (list$175) **$125**
Blaise **Turbo C Tools** for Turbo C (list $129) ..... **$95** *New!*
Blaise **Turbo Power Tools +** (list $99) **$79**
Borland **Turbo Basic** (list $99) **$61**
Borland **Turbo C** (list $99) **$59**
Bytel **Genifer** dbase III application generator ($395) **$225**
Comtel **dbScope** Interactive dBase debugger ($70) **$62**
Comtel **dbTools** (list $70) **$62**
Creative **Vitamin C** Specify compiler (list $225) **$150**
Creative Programming **VC Screen** (list $100) **$79**
Digitalk **Smalltalk/V** (list $99) **$84**
Fox **FoxBase + v2** fast dBase compiler ($395) ..... **$249** *New!*
Gimpel **PC-Lint** (list $139) **$95**
Matrix **Synergy Development Toolkit** (list $395) .. **$309** *New!*
Macintosh-like icons, menus, multi-tasking, mouse, more!
Metagraphics **Metawindow/All Language** ($195) .. **$159**
Windows, graphics, mouse support
Metagraphics **Metawindow/Turbo C** ($95) **$75**
Metagraphics **Metawindow/Turbo Pascal** ($95) **$79**
MicroHelp **Mach2 for Turbo Pascal** ($69) **$55**
windows, fast i/o, critical error trapping, more
Microsoft **C v5** w/Codeview & Quick C (list $450) ..... **$279** *New!*
Microsoft **Quick C** (list $99) ..... **$68** *New!*
Microsoft **Fortran** w/Codeview (list $450) **$270**
Microsoft **Quick Basic v3** (list $99) ..... **$45** *Save!*
You pay TSF $65 and get $20 rebate from Microsoft
MKS **Toolkit** Korn shell, vi, grep, 30 more ($139) **$115**
Periscope **Periscope II** (list $175) **$139**
Phoenix **Plink-86 +** (list $495) **$315**
Softfocus **"C" BTree & ISAM** w/source ($115) **$99**
Sterling Castle **Basic Development Tools** ($99) **$79**
B-Tree & windows w/source for MS & Borland basics
Sterling Castle **Blackstar "C" Functions** ($129) ..... **$99** *New!*
Full featured library w/source for both MS & Borland
Texas Instruments **PC Scheme** great AI into ($99) **$77**
Tom Rettig **Library** specify dBase or Clipper ($99) **$79**
Turbo Power **Extender** (list $85) **$68**
Turbo Power **Optimizer** (list $75) **$59**
Turbo Power **Optimizer** w/Source (list $175) **$99**
Turbo Power **TDebug Plus v2** (list $60) **$48**
Turbo Power **Turbo Utilities** w/source ($95) **$76**
Unipress **Phact** RDBMS ($249) ..... **$199** *New!*
includes report writer, query program & all source
Wallsoft **UI Programmer** (list $295) **$260**

TSF carries hundreds of products from dozens of publishers. Call or write for a FREE catalog or a price quote.

Number **98** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Total Confusion

I have followed Michael Abrash's articles on the EGA with great interest because I am currently preparing a graphics module for our Finally! product line. Mr. Abrash failed to mention that the current state of affairs in the VGA/EEGA market is total confusion.

Each manufacturer currently has a different method of implementing its 640 x 480 mode. This is due in part to IBM's delay in defining an industry standard, which they have finally done with the VGA modes. These now include mode 11 (640 x 480 with 2 colors), mode 12 (640 x 480 with 16 colors), and mode 13 (320 x 200 with 256 colors). Mode 12 is comparable to what is now called the EEGA mode. Each mode is called by an interrupt 10.

As far as I have been able to determine, other manufacturers use different modes for their 640 x 480 EEGA modes. Tseng Labs/NEC use a mode 25 call to int10, Paradise uses mode 50, STB uses mode 9, and Sigma uses mode 7. The Vega/Quadram card is set totally by software commands to the EGA processor.

Add to this the proliferation of other resolutions, each with its own initialization sequences. Sometimes I feel as if I buy video cards just so that the manufacturer can make them outdated! It would be nice if the video card manufacturers had an update policy similar to software manufacturers'—"Just mail in your old card and we'll send your updated *Super-rest++multimode/deluxe\autocard* for $. . . ."

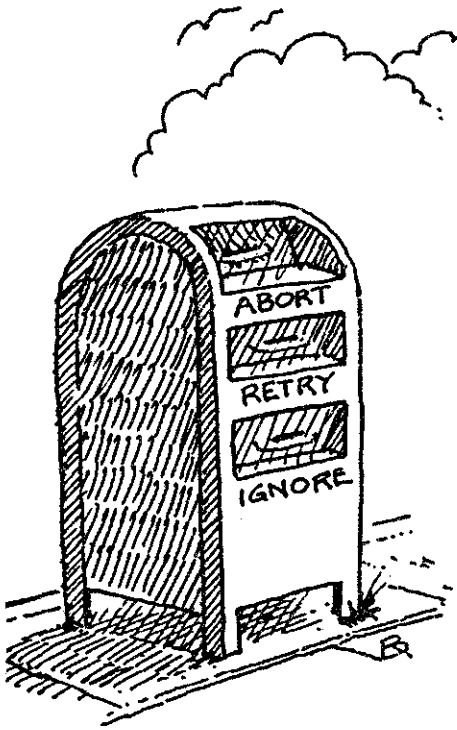
Paul P. Dymerski
Richmond, VA

More on the EGA

I have been reading your EGA articles in *PJ* with interest. I've been doing some minor hacking on the thing, mostly in a vacuum. My most recent prior experience has been on an Amiga, and in spite of the EGA's sophistication, its limits are painful to an AmiGuy.

Let me comment on some notes I made while reading your articles and then pose a question or two. I'll refer to the article in March/April 1987 (Volume 5.2) as Part 1 and to the article in May/June 1987 (Volume 5.3) as Part 2.

In Part 1, page 42, you pose the following question: What benefit is the OR mode of the EGA ALUs? Unless I'm mistaken, this is the mode you'd use to copy an image into the screen with portions of the image transparent. Because all images are not rectangular, this is very useful in an animation sequence. True, you could vary the mask register as you write each memory location, but that is a performance hit. (Pixel value zero is transparent in OR mode.)



You made mention of trying to write an optimized Bresenham, which I read on the same day I had been pondering the problem. I, too, figured to keep a rotating mask with a single bit set. The mask is cumulatively OR'ed with a byte mask that represents the pixels occurring in a single screen buffer byte. When the "end of a byte" is reached, the ega bit mask register is set to the cumulative mask, and 0xFF is written to the EGA. Indeed, rotating the bit out of the mask (in your AL) calls for incrementing the screen address (your DI), but you must do more than that. Adding with carry only handles DI, but since you need to test the rotating mask anyway (the cumulative mask must be cleared; a write to the screen must take place), and ADC presumably takes longer than "INC DI," the marvelous trick you report has little use, I think. This might not be the case in your usage, of course.

I sketched out an assembler version that does Bresenham completely in 8088 registers (except program fetch and writes to display). It uses only a minimum of self-modifying code in the setup, so I will be only a minor sinner if I get the time to code it up (the C version is fast enough for my immediate needs). How did your version turn out?

As for the general topic of calculating timings of combinations of instructions, if we ever get our state analyzer working with the Deskpros, we can look right at the timing of the executing code. I assume that would answer all questions. It seems that my Deskpro is too high-strung to tolerate the analyzer for long. The analyzer will certainly tell you if the second byte of a branch instruction is not fetched if not needed (odds are slim).

Some general EGA comments:

We found it difficult to obtain EGA reference manuals. We found it a lot easier to get the manual from Compaq, which is presumably equivalent, is laid out nicely, and has some useful example code. Some information on EGA reference material should be included in every EGA magazine article, in my view.

I am not too thrilled with the gyrations one must go through to use EGA with a Microsoft mouse or equivalent. One uses extensions of the BIOS (int 10h!) to set registers so that the mouse driver can change/restore them without reading them (they are write-only).

My first impression is that the fact that the Bit Mask register is not subject to the barrel shifter is stupid. I guess that to place a character (or some other image), you set the Bit Mask once (rotate it yourself with the CPU) and write a larger set of data that goes through the barrel shifter, but it is still stupid.

Another annoyance: It is possible to set up the EGA so that a write of a byte to the screen buffer can set one pixel value corresponding to the set bits and another pixel value for the reset bits (so that you can make, for example, a dashed horizontal line of two colors), but the selection of the pixel values is severely limited!! The GCD registers involved are Enable Bit Set/Reset and Data Set/Reset. The former determines whether values in specific bitplanes will be forced to a value; the latter determines the value to which they will be forced.

You can thus arrange to have a one bit in a data byte cause pixel value 1110 with a zero causing 0010 by setting EBS/R to 0011 and DS/R to 0010. You cannot, however, have a one-bit cause 0011 and a zero-bit be 1100. The encoding of these registers could have supported such a thing by assigning meaning to the ignored bits of DS/R. The meaning would be: if 0, data is as provided by processor; if 1, data is complement of that provided by processor. EBR/S of 0000 and DS/R of 1100 would then accomplish my desired goal of arbitrary "foreground/background pens."

My question(s):

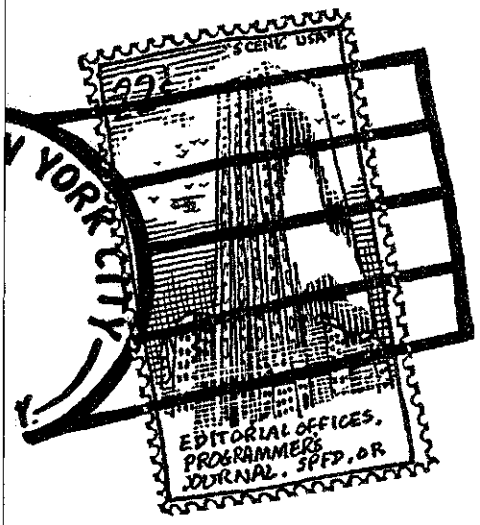
You made reference to an abhorrent but common practice of scanning the EGA BIOS for "IBM" to determine if an EGA is around. My needs are to determine which graphics mode to use. I was surprised to find that the equipment list (at 0040:0010) didn't abide by the Compaq manual and insists that I have an 80 column CGA—even with the Compaq EGA installed. The best idea I have so far is to try to set the display mode to my acceptable values, in succession, testing each time if the mode change "took" (by checking the current video mode after I have attempted to change it). My

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

order is: 10h, color ega; 0fh, monochrome ega; 6, CGA graphics; and so on.

This will be okay (if it works out) because I want to go into graphics mode right off the bat and stay there and I will stop changing modes as soon as one works. But it, too, is pretty unsatisfying. I guess it's "welcome back to the wonderful world of the PC again."

Jim Mackraz
Santa Clara, CA
bix: jmackraz
usenet: mitsumi!jimm <to appear>



Michael Abrash Replies

Jim,

Lord knows, the EGA IS painfully limited as compared to an Amiga.

The OR mode of the EGA's ALUs can be used for certain types of image transparency, but it's really not very well suited to it. The problem is this: The image being copied in is, presumably, in a certain color (say red, for this example). In order to guarantee it will appear correctly where it overlaps with other images, planes 0, 1, and 3 must be forced to 0 (otherwise, odd color effects will result from the combination of the new red image and old image data in other planes). The OR mode provides no way to force selected planes to 0 where the new image data is 1 while leaving them unchanged where the new image is 0. (This is similar to your later—valid—complaint about the limitations of the set/reset circuitry.) The closest approximation would be to use the AND mode with a mask to clear the image space and then use the OR mode to insert the image, which is indeed a benefit of the OR mode. I think that the OR mode by itself would be useful only in a single-plane (two-color) context, where color interference would not be a problem.

I'd like to hear what you find with your state analyzer, but I'm not convinced it will answer all questions. The problem is that the analyzer can monitor only bus activity (I think); there's no way of telling exactly when a given instruction begins and ends inside the processor.

As I write, the current issue of *PC Tech Journal* (I think it's July) has an article by Augie Hansen that describes how to identify the display adapters installed in a given system. I haven't tested the code, but Augie does good work. By the way, there are good historical reasons why the equipment byte at 40:10 insists you have an 80-column CGA; otherwise, software written before the EGA existed wouldn't know that there was a color adapter installed. Yes, I admit that the world of the PC is a bit of a mess, but there are 10,000,000 or so PCs out there, so it's a mess worth working with.

I didn't know that Turbo C was based on Wizard C (still don't, for that matter; I've heard it's written by the same person, but I don't know how closely related the two compilers are). I have no idea how Wizard C stacks up. It is certainly true that most C compilers do multiple passes and so should have at least the temp files and preferably the compiler itself (but not the source files for any non-masochist!) on a RAM disk for best performance. In my experience, Turbo C is about 4 times as fast as Microsoft C running with everything on a RAM disk. Finally, Turbo C does not insure there is a copy of the source on disk before running a program from the user interface environment although it does prompt the user if the source has not been saved when exiting.

Thanks for the excellent feedback.

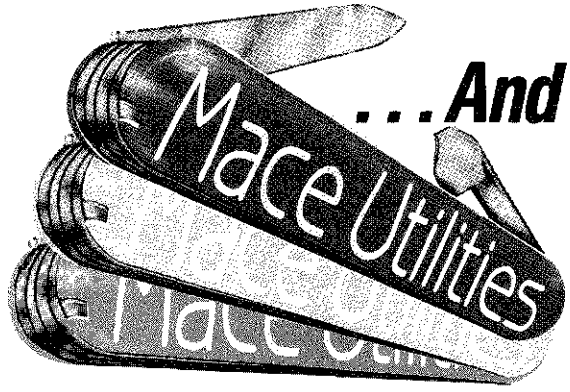
Michael Abrash
Mountain View, CA

Continued . . .

"Mace Utilities 4.0 Best Data Recovery There is Gets Better"

PC Magazine
March 31, 1987

. . . And Better!



VERSION 4.10

RECOVERY FEATURES

DIAGNOSE checks any disk for errors.
REMEDY fixes errors, automatically.
UnDELETE recovers deleted files easily, automatically.
UnFORMAT recovers from an accidental format of your hard disk even when MACE was not installed.
new *Floppy* **UnFORMAT** recovers from accidental format of a floppy disk.
new ***FORMAT-F*** floppy formatter. Replaces DOS FORMAT. Includes **'RESURRECT'** option, to revitalize weary disks.
new ***FORMAT-H*** hard disk formatter replaces DOS FORMAT. Guaranteed safe.

HOT-ROD FEATURES

new *Fast* ***UnFRAGMENT*** quickly reorganizes segmented files for speedier disk performance.
VCACHE buffers data reads and writes in regular, expanded or extended memory for dramatic disk speedup.
new ***SORTD*** Directory sort.
new ***SQZD*** Directory path optimizer.
VSCREEN Screen speedup.
VKETTE Diskette caching for floppies.
new ***dbFIX***
Repairs and recovers damaged or lost DBase files, automatically.

Advanced performance at the same good price: $99

To Order Call
800-523-0258
503-488-0224

Paul Mace
SOFTWARE
400 Williamson Way, Ashland, OR 97520

Number **80** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Use the "NUL" File

I'd like to suggest a small improvement to the ideas presented by Paul Dymerski in his article "Formatting BASIC Output with Buffered PRINT USING" in the May/June 1987, issue (Volume 5.3) of *Programmer's Journal*.

The idea of using a file buffer to retrieve the results of a PRINT USING statement is both useful and clever. The process can be made simpler and faster if the "NUL" file is used for this purpose. This character device, which throws away its input, can be opened as a random access file from both the BASIC interpreter and QuickBASIC.

There are several benefits to using "NUL" as described in the article. First, it avoids the delay of opening a real disk file. Second, it eliminates the danger of the user's changing floppy disks while the program is formatting its output. Third, it eliminates the need to erase the file when the program is finished with it.

John Bresnahan
Minocqua, WI

Turbo C Follow-up

When I previewed Turbo C, I promised to let readers know how my experience with Turbo C turned out. I understand that you're going to print a full review by Frank Greco soon, so I'll just update my review. (I suspect that many of *PJ*'s readers already know Turbo C firsthand since over 60,000 orders have reportedly been placed for it.) I've used Turbo C pretty heavily over the last three months; among other things, I've ported a program with 2 megs of C source and ½ meg of assembler source from Microsoft C 4.0 to Turbo C. The port took about three days, and while it was certainly not trivial, much of the difficulty was a direct result of the complexity of the program and of the then-beta status of Turbo C. If you're considering porting, I'd say it may take some effort if you've got a lot of assembler and/or static far data in a near data model. I must add, however, that in my case the port was *well* worth it.

Turbo C is very much as Borland claimed it would be. Most importantly, it does indeed compile and link like a bat out of hell. The compiler is about 4 times faster than MSC even when MSC has all the compiler phases and temporary files on a RAM disk, and the speed advantage jumps when a RAM disk isn't used. The linker is much, much faster than MSLINK, although I haven't timed the difference.

Code size was about one-half percent smaller than MSC for my application. Code speed is frankly impossible to judge since most of my time-critical code was assembler.

While Turbo C does produce standard OBJ files, they are perhaps too standard; they're Intel standard, which is not exactly the same as Microsoft's OBJ format. Microsoft uses comment records to pass information to the linker; since neither Turbo C nor Borland's linker uses comment records in this way, there are problems using MSLINK with Turbo C and Borland's linker with MSC. Once I got Borland's linker working, though, I didn't much care if I ever saw MSLINK again; Borland's linker is fast and unbelievably compact (less than 10K!).

Inline assembler code works very well; however, the way it works is that when inline assembler is used, Turbo C compiles to assembly code, then invokes MASM to assemble the code. This is a slower process than direct compilation, and does require MASM. Often, however, inline assembler turns out not to be needed: for example, to set the video mode, I originally used in-line assembler thusly:

```
asm mov ax,[VideoMode]
asm int 10h
```

but found that direct access to registers in C and geninterrupt(int_number) allowed me to replace that code with:

```
_AX = VideoMode;
geninterrupt(0x10);
```

Turbo C provides a number of other features, such as far heap management in near data models, that make assembler less necessary.

I've used Turbo C's self-contained user interface environment, and it's not bad at all. In fact, when I'm compiling a lot of new code, I generally use the user interface because it's the quickest way to identify and fix code with several errors. The editor's quick and functional although it is not as powerful as the best editors and is WordStar-like. Still and all, I prefer to use Brief for most work. . . but then, that's why Turbo C has a command-line interface too.

Things I haven't exercised yet and hence can't comment on: floating point and interrupt type functions.

Finally, Turbo C lists for $99, and I've seen it available for as little as $59. Library source was originally priced at $295, but I've just read that the price has been cut to $150.

In short, Turbo C is as good a value as Borland claimed. It's now my C of choice, and the only complaint I have is that it doesn't come with a debugger. (SYMDEB works with it, but to get that you have to buy MASM.) Of course, the world doesn't stand still, and Microsoft has announced both Quick C (at $99 dollars, it includes a user interface environment and a source-level debugger and is reported to compile even faster than Turbo C) and C 5.0 (claimed to produce 30% faster code than C 4.0). Microcomputers and competition together are a wonderful sight to behold. . .

Michael Abrash
Mountain View, CA □

Tom Rettig's Library™

Prewritten Solutions to Programming Problems

Clipper Edition

Advanced Extended Library

- Convenient .LIB library file to link
- Separate object files so only the code actually used is added to your program
- Optimized for speed and code size
- Requires Clipper, Winter '85 or later

dBASE III PLUS Edition

Advanced Programmer's Library

- Interface to C/assembler is fully compatible with Clipper's Extend system
- Functions are CALL procedures
- Requires dBASE III PLUS, any version, and 128K of additional memory

Each edition comes complete with...

- Entire source code, nothing withheld. Cleanly written, liberally commented, easy to modify, helpful for learning.
- Support by phone and electronic mail
- No royalties or copy protection
- Documentation inserts for dBASE or Clipper manual. Accurate, complete, easy to use, page-per-command.
- Handy plastic reference card
- Full money-back guarantee

Over 175 functions; 55% written in C, 35% in Assembler, 10% in dBASE

$99.95 per edition at dBASE/Clipper dealers or direct

II/III Tom Rettig Associates
Excellence in dBASE since 1982

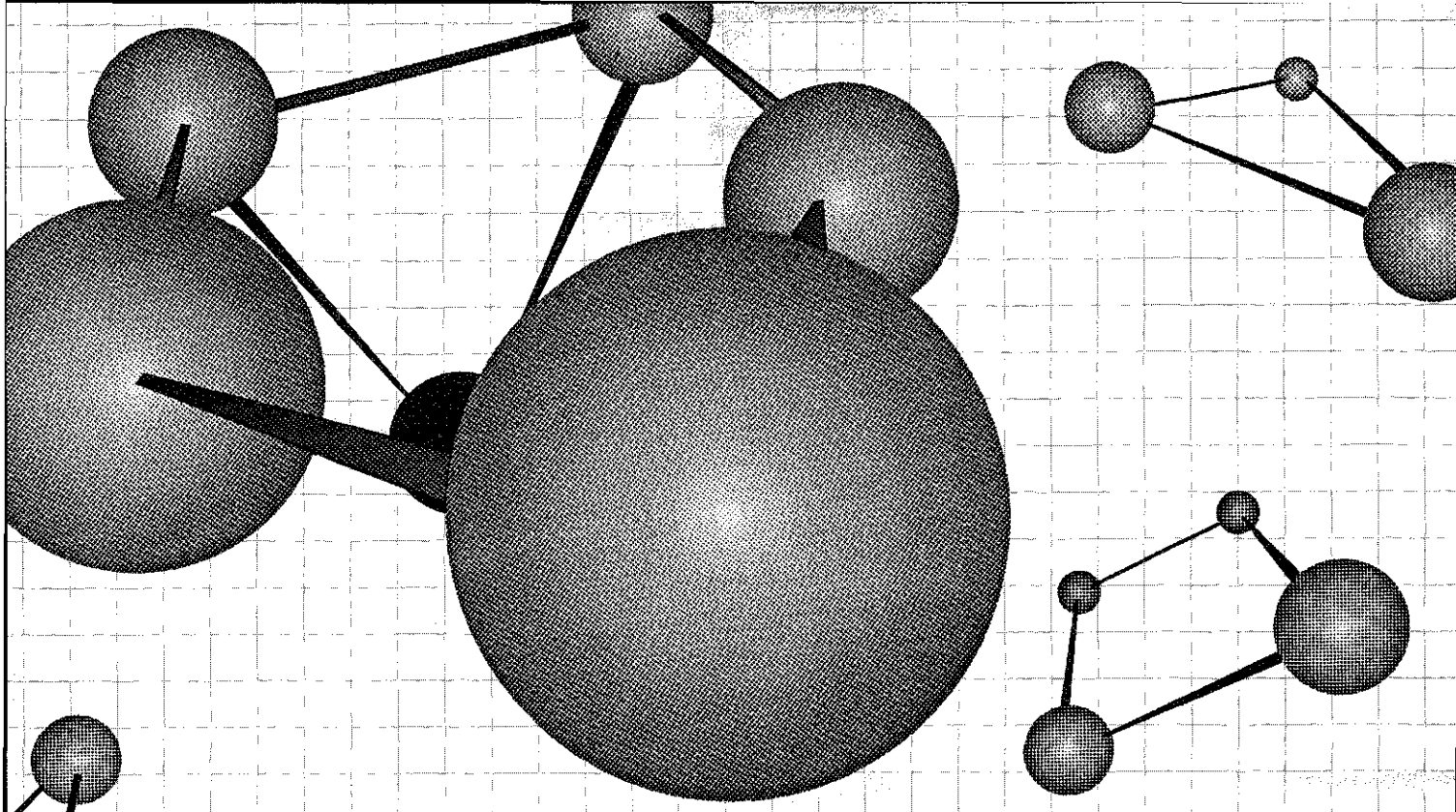
9300 Wilshire Boulevard, Suite 470
Beverly Hills, CA 90212-3237, U.S.A.
(213) 272-3784 ■ Telex: 4996426 RETTIG
Source: BCR480 ■ CompuServe: 75066,352

dBASE and dBASE III PLUS are Ashton-Tate trademarks, Clipper is a Nantucket trademark

Call or write for free product information

Number **117** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Aztec C - the *best* C in the Micro-cosm

Why is Manx Aztec C the best C ?

Because Manx offers a combination of performance, features, and pricing that are unbeatable.

What kind of performance ? In the 2/87 issue of Computer Languages, our 3.2 release of Aztec C for PC-DOS turned out 109,489 iterations of the Dhrystone benchmark per minute to108,303 for Microsoft C. Our 4.1 code is even faster.

What kind of features ? Features like in-line assembly code, a professional make utility, library source, ROM support, debuggers, assemblers, and overlay linkers. Many of these features are unavailable or extremely high priced options in other systems. Microsoft, for instance, does not provide library source or in-line assembly.

What kind of pricing ? For $199, the Professional version of Aztec C gives you an Optimized C, a macro assembler, a librarian, overlay linker, extensive UNIX and MS-DOS specific library routines, a powerful debugger, sample programs, and over 500 pages of documentation. Manx Aztec C has a hard earned and well deserved reputation as a leading C software development system for PC-DOS and MS-DOS. At this price, it is also one of the best software values available.

Over 80 percent of our sales are repeat sales or are based on recommendations. That says it all. At Manx, we extend ourselves to the limit to provide the best software, the best service, and the best support possible. Call today, and join the over 50,000 users who know the Manx difference.

Manx also provides C systems for Macintosh, Amiga, CP/M-80, TRS-80, Apple //, 6502/65C02 ROM, 8080/Z80 ROM, 680x0 ROM, and 8086/80x86 ROM. **Call for prices and information.**

Aztec C86 4.1

PC-DOS • MS-DOS • ROM

Aztec C86-p Professional System $199

• optimized C with near, far, huge, small, and large memory • in-line assembler • in-line 8087/80287 • ANSI support • Fast Float (32 bit) • optimization options • Manx Aztec 8086/80x86 macro assembler • Aztec overlay linker (large/small model) • source level *debugger* • *object librarian* • MS-DOS 3.x *file sharing & locking* • comprehensive libraries of UNIX, MS-DOS, Screen, Graphics, and special run time routines.

Aztec C86-d Developer System.....$299

• includes all of Aztec C86-p • Unix utilities make, diff,grep • vi editor • 6+ memory models • Profiler.

Aztec C86-c Commercial System.....$499

• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 cross dev. option • One year of updates.

How To Become An Aztec C User

To become an Aztec C user call **800-221-0440.** From NJ or international locations call **201-542-2121. Telex: 4995812** or **FAX: 201-542-8386. C.O.D., VISA, Master Card, American Express, wire (domestic and international), and terms are available.**

Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee. Systems are upgradable by paying the difference in price plus $10. One and two day delivery is available for all domestic and most international destinations.

MANX

Manx Software Systems
1 Industrial Way, Eatontown, NJ 07724

To order or for more information call today.

1-800-221-0440

In NJ or international call (201) 542-2121 • TELEX: 4995812

REPRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Slouching towards the 386:

The Intel 80386 Processor versus IBM's Operating System/2

by
Christian Walker

The Second Coming

Turning and turning in the widening gyre
The falcon cannot hear the falconer;
Things fall apart; the centre cannot hold;
Mere anarchy is loosed upon the world,
The blood-dimmed tide is loosed, and everywhere
The ceremony of innocence is drowned;
The best lack all conviction, while the worst
Are full of passionate intensity.

Surely some revelation is at hand;
Surely the Second Coming is at hand.
The Second Coming! Hardly are those words out
When a vast image out of Spiritus Mundi
Troubles my sight: somewhere in sands of the desert
A shape with lion body and the head of a man,
A gaze blank and pitiless as the sun,
Is moving its slow thighs, while all about it
Reel shadows of the indignant desert birds.
The darkness drops again; but now I know
That twenty centuries of stony sleep
Were vexed to nightmare by a rocking cradle,
And what rough beast, its hour come round at last,
Slouches towards Bethlehem to be born?

—William Butler Yeats

The study of the instruction set of a family of microprocessors could seem like an interesting topic for a college paper, but would seem to have little bearing on the practitioner. After all, most programmers use high level languages, or languages such as C, to program. With luck, these languages should hide the messy facts of life of the instruction set.

But programmers of the Intel microprocessors know better. The form of pointers in C, for instance, changes depending on the memory model used. The language system might have arbitrary limits on total data or code space used. Or performance might suffer if data types that are not supported with the hardware are used. The fact is that the instruction set is *not* invisible, and programmers can't help but be affected by the processors they use.

But what is the point of learning about instruction sets of processors that you don't program? If you are an IBM PC programmer, all you need to know is the Intel 8088 instruction set, right? Wrong. Hardly a week goes by without the announcement of another machine that uses either the 80286 or the 80386.

The future rides on these new processors, and the higher speeds of these machines is not the only driving force. Hardware support for large address spaces, memory management, and memory protection are just a few of the features included in the new processors.

One possible way to make sure you are developing for the future is to work on the newest machines. But this doesn't always work. Intel produced the 80186/80188 processor before the 80286. This processor was faster than the 8086 and had some extra instructions. But it was never very popular. You might have produced a great product for an unpopular machine.

Or you might just decide that anything IBM makes will be popular. Using this rule of thumb, you might have produced great products for the PCjr. Even giants such as IBM can't predict the future in this business. And picking the right processor is only part of the solution. The right operating system to run under can be critical to your application. If your operating system does not support programs larger than 64K bytes, it may not run your program.

On the assumption that the knowledge of history will better prepare us for the future (or present), this article will explore the Intel processors, beginning with the 8080 up through the 80386, and finally venture into IBM's latest operating system. I will concentrate on the application-programmer-visible part of these machines; life is too short to cover circuit designs or system designs.

The Intel 8080

The history of the Intel instruction set actually starts in 1971 with the 4004 4-bit processor, which was the *first* microprocessor. Used for applications such as decimal calculators, these first micros were slow and had very limited memory. A typical instruction could add two 4-bit BCD numbers.

The 8008 followed in 1972, but it was the 8080, introduced in 1974, that had a large impact. Before the 8080, microprocessors were used for small, embedded, preprogrammed applications such as handheld calculators, microwave ovens, etc. But the 8080 opened up a whole new world to the general computing community.

The 8080 had built-in support for 8- and 16-bit integers and instructions to handle 4-bit BCD arithmetic. The BCD support was left over from the 4004 and, as we will see, still exists on the latest processors from Intel. A number of other 8080 features have also carried over into the newer processors, but not directly: for example, the jump instruction with 16-bit offsets was present in both the conditional and unconditional forms, but on the 8086 only the unconditional jump had a 16-bit offset—conditional forms had an 8-bit form only. The processor had a stack pointer register, and the CALL instruction used it to save the return address. This easy use of a stack did not help languages that use stack frames (C or Pascal, for instance), but it was useful for ROMable systems since code was never modified as it is on some machines during a CALL instruction.

There were a few instructions for dealing with the stack. PUSH and POP, as well as an instruction to exchange a register with the top of the stack, also existed. Programmed input/output was done with the IN and OUT instructions; vectored interrupts existed in a primitive form; and there was a way to enable and disable interrupts. Addresses were 16 bits, which allowed 64K bytes to be addressed.

This machine was not oriented to high level languages. Even though it had a stack, a stack frame pointer register—so important for languages such as C—was missing.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

And every register had different uses and functions. The code emission section of compilers for this machine must have been interesting at best. But for an assembly language programmer, the machine wasn't too bad. It had the conditional CALL instruction as well as fast CALLs to up to eight routines. Still, it was lacking complete support for multi-precision arithmetic and strings. And it had only one register that could be used as a general index register.

There was also an important omission that still affects us today: The 8080 had no multiply and divide instructions. It was easy to write subroutines to accomplish the operation with shifts and adds, but this resulted in much slower operations than those done directly in hardware. Programmers found ways to avoid the multiply and divide by changing the algorithms used—the famous SIEVE benchmark program was written for such an environment. And some compilers were programmed to optimize multiplies by small constants.

In fact, for general computing, this processor was unsafe, slow, and hard to deal with. To make it easier to deal with, some higher level safe languages, such as UCSD Pascal, were implemented. Bill Gates' first BASIC interpreter came to life on this machine, and, though it made the machine even slower, it did improve safety and productivity.

The next processor from Intel was the 8085, available in 1976. It added only two instructions: SIM and RIM, which controlled interrupt handling. But the processor could still address only 64K bytes of memory. This processor was successful, but not as successful as its competitor, the Zilog Z-80.

The Zilog Z-80

No history of Intel processors would be complete without mentioning the Zilog Z-80 chip. This processor could run 8080 programs, plus it had several features that made it extremely popular. Zilog added two index registers, a set of shadow registers useful for fast handling of interrupts, fairly complete support for multi-precision arithmetic, bit instructions, more flexible I/O control, and fast string handling—all in a package that reduced chip count in a complete computer system.

It is interesting to note that Zilog used different assembler mnemonics in its manuals. Intel copyrighted the mnemonics for the 8080, and Zilog decided not to challenge Intel's "right" to use such words as "ACI," "STA," or "XCHG." At the time Intel accepted the idea that separately engineered processors could work on binary instructions defined by Intel, but as the standard maker, they wanted to "protect" something of their work. So they picked the assembly language definition. Zilog easily got around the restriction, but now programmers have to remember two separate sets of mnemonics.

The Intel 8086

In 1978, Intel produced the true successor to the 8080: the 8086. The 8088, the 8-bit form of this 16-bit processor, was available in 1979. This processor was not compatible with binary programs from the 8080, but most 8080 programs could be reassembled to run on the 8086. This form of compatibility with older processors is unique to Intel. At the time, most other microcomputer companies were satisfied with just inventing new and improved instruction sets and processors. But Intel was already worried about the existing customer base and the existing software base.

This concern did not go unrewarded. Since the 8086/8088 was similar to the popular 8080 and Z-80 processors, microcomputer software was easily ported to it. By the time IBM was looking for a processor to put into a microcomputer, Intel had a machine with (1) growth potential and (2) *software* while its competitors were still promising only one of these two.

The 8086/8088 processor retained most of the functionality of the old 8080 processor. Some of the more obscure instructions were not implemented, but string operations were supported with the repeat instruction and memory-to-memory instructions. Multi-precision arithmetic was supported with the ADC and SBB instructions, and the Multiply and Divide instructions were finally added. 16-bit index registers were added to the register set. One of these index registers, BP, was specially tailored to act as a stack frame pointer, to better support languages such as C and Pascal.

Intel also added segment registers to the register set. Although addresses were only 16 bits, these registers allowed total memory size to be larger than 64K bytes. Every instruction that addressed memory had a segment register that was included in the address calculation. The 16-bit offset that was calculated using index registers and constant offsets (modulo 64K) was added to 16 times the segment register value to get the resulting 20-bit address that the chip sent to the memory. This address was not visible to the program, and there were no instructions that could use a 20-bit number as an address.

Programs could be unaware of the segment registers if they used only 16-bit addresses. All instructions had defaults for the segment registers that were involved in the effective address calculation. With luck, the operating system would set up all the segment registers to be the same value.

Programs with just a bit more smarts could enjoy separate address and data spaces, each 64K bytes, without undue program changes. These programs had to have separate code and data segments. In most cases, this was easy because the 8080 instruction set was designed with read-only memory in mind. All of the Intel processors made it easy to write pure code, that is, code that is not written to. This feature was no mistake; it allowed code to be put into ROM for those applications that needed it.

The fact is that the instruction set is not *invisible, and programmers can't help but be affected by the processors they use.*

So it was easy to access 64K bytes of code and 64K bytes of data at one time on the 8086. But to address larger areas, the machine code had to manipulate the segment registers. This was not easy or fast for a number of reasons. The segment registers could only be loaded, not added to or subtracted from. There were no instructions to convert a 20-bit address to a segment-offset pair. Loading a segment register and index register was supported with instructions, but indexing from one of these pointers could be done with 16-bit quantities only. Objects larger than 64K bytes were not supported in the instruction set and required gobs of code to access.

Continued . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

With the 8086 Intel also added support for co-processors. The interface to the co-processor was general to allow many different types of co-processors, but, to my knowledge, Intel produced only one for the 8086: the 8087 numeric data processor. The 8087 handled floating point numbers with instructions such as add, subtract, multiply, divide, and square root plus exotic instructions to help calculate sines, tangents, arctangents, etc.

Other processors co-existed with the 8086, but typically not in the 8087's close relationship. Intel had produced I/O processors such as the 8089 I/O processor, graphics processors, EtherNet processors, etc. These processors, however, could be used with other microprocessors like the (God forgive) Motorola 68000. There seems to be quite a PR campaign going on to get companies to adopt one of the various graphics processors. We can only hope that all this competition will at least result in more useful, cheaper products in the near future.

Intel also added some features to help programmers debug systems. There was a way to single-step through machine instructions and a one-byte instruction (INT 3) that could be used by a debugger to set breakpoints; however, all bit patterns would execute and do something on this processor since it had no idea of privileged or illegal instructions.

The Intel 8088, 80186 and 80188

In the 8088, the 8-bit version of the 8086, the instruction set was the same, but the chip itself had only 8-bit data paths. So loading a 16-bit word from memory took two memory fetches. The advantage of this form of the 8086 was reduced cost of memory and smaller memories.

The original IBM PC, which used the 8088, was sold with 16K by 1-bit RAMs. This allowed IBM to sell a machine with as little as 16K bytes of RAM. In 1981, it was important to offer a machine with such small memory. If the 8086 had been used, the minimum system would have been 32K bytes, and bigger bus connectors and bus driver would have added more expense.

In 1982, Intel brought out the 80186 and 80188 processors—with some speed advantages over the 8086 and a few extra instructions. But this processor's major advantage was that it reduced the chip count of complete computer systems by integrating clock, timer, and DMA circuitry on the chip. This processor finally recognized illegal opcodes and illegal operations and interrupted the program when they were encountered.

The instructions added were useful for high level languages. The BOUND instruction allowed subscript checks with little time or space penalties. The ENTER and LEAVE procedure instructions were added to help set and restore the BP register used in high level languages. They could also move displays to the new stack frames. This feature could be useful for Pascal, which uses displays, but to date, I know of no compiler that uses this complex instruction—probably because it wouldn't work on the standard IBM PC.

Other minor improvements were offered with the 80186: immediate PUSH and multiply instructions; block I/O instructions; shifts by small constants; and pushing and popping all registers with one instruction. Despite these improvements, the 80186 and its 8-bit brother chip, the 80188, were never popular for general purpose computers. They were used most on controller boards, largely because they were fast and allowed a lower chip count.

Intermission

So by 1982, Intel had processors that supported 8- and 16-bit integers, "high level" languages like C, multi-precision arithmetic, and up to a megabyte of memory with 64K byte objects. The optional co-processor supported floating point numbers. Obviously missing were support for virtual memory operating systems, memory protection, objects larger than 64K bytes, and address spaces larger than 1 megabyte.

The success of a processor in the 1980s and 1990s had better not be based on its interest to assembly language programmers.

There were two kinds of pointers on these processors, and they were quite visible to the application program. This was basically unacceptable, and the competition was moving to solve the problem. The Motorola 68000 family had started shipping by 1980 and boasted full 32-bit registers with at least 24-bit addresses. This 16 megabytes of *linear* address space was and still is a major advantage over the Intel processor's addressing.

Motorola also touted its regular instruction set and general purpose register set. Intel was stuck with an irregular instruction set and, probably more damaging, a special purpose register set. There was not one register in the Intel 8086 that did not have at least one special property.

As a result, code generators for the Intel machines could not be written easily, and the users of these compilers have to put up with either (1) slow code, (2) buggy code, or (3) fast code from large compilers that took years to write and debug. To this day, we are seeing new releases of C compilers that claim to emit faster code than last year's compilers. This activity is indicative of a machine that's hard to deal with.

My gentle reader should not conclude that I do not like the Intel processors. Writing assembly language programs is great fun; it's challenging and full of opportunities for optimization hard to imagine a compiler could do. But to write optimized programs for these machines, one needs the Intel manual with timings always handy. And, on the IBM PC, the Intel timings have to be modified with the knowledge that instruction fetches are not free.

In fact, I find that the most interesting machines to program in assembly language are those that are irregular, and the most boring ones are those that have regular instruction sets and lots of similar registers. But the success of a processor in the 1980s and 1990s had better not be based on its interest to assembly language programmers. The successful processor should be easy for compilers to emit code for and have support for larger and larger programs and data.

The Intel 80286

So Intel responded to market pressure with the 80286 processor, available in 1982. The 80286 supported 24-bit addressing, memory protection, and virtual memory. And it even allowed "old" 8086 programs to be run. But because 8086 programs could address only 1 megabyte of memory, not 16 megabytes, the ground rules for effective address calculation were changed.

The segment registers were no longer multiplied by 16 before being added to the offset. Instead, segment registers were an index into an operating system table that contained base pointers and protection information. A selected base pointer was added to the instruction's 16-bit offset to get the required 24-bit address.

This table was not accessed for each effective address calculation, but only when a segment register was loaded. With this scheme, the operating system set up the mapping between real memory and the program's addresses. Two segments could point to the same memory, or two segments could overlap. If the table were properly set up, the system could duplicate the 8086 addressing, but this required

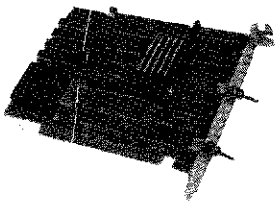
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MICROWAY ACCELERATES YOUR PC!

FastCACHE-286™

Runs your PC Faster than an AT!

Runs the 80286 at 9 or 12 MHz and the 80287 at 8, 9 or 12 MHz. Includes 8 kbytes of 55ns CACHE.



Compatible with Leading Edge Model D, Compaq, and Turbo motherboards. Includes 8088 Reboot Switch, DCache, Print Spooler and Diagnostics... **From $399**

LOTUS/INTEL EMS SPECIFICATION BOARDS

MegaPage™ The only EMS board which comes populated with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache and EMS drivers. For the IBM PC, XT and compatibles...**$549**

MegaPage with 0K..... **$149**

MegaPage with 2 megabytes of HMOS RAM..... **$419**

MegaPage AT/ECC™ EMS card for the PC AT and compatibles includes Error Correction Circuitry. With ECC, 11 RAM chips cover 256K so the user never encounters RAM errors. With 1 megabyte CMOS RAM..... **$699**

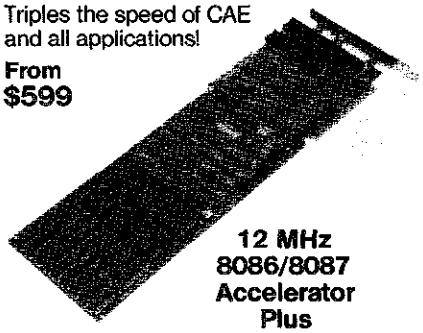
INTEL, JRAM, or Maynard ..... **CALL**

INTEL INBOARD 386 0K..... **$1325**

NUMBER SMASHER/ECM™

Triples the speed of CAE and all applications!

From $599



12 MHz 8086/8087 Accelerator Plus A Megabyte for DOS!

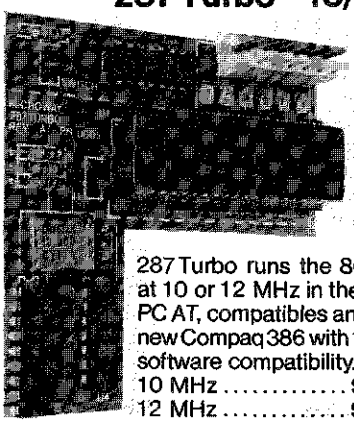
For the IBM PC, XT and compatibles

***PC Magazine* "Editor's Choice"**

8087 SOFTWARE

IBM BASIC COMPILER	$465
MICROSOFT QUICK BASIC	$79
87BASIC COMPILER PATCH	$150
87BASIC/INLINE	$200
IBM MACRO ASSEMBLER	$155
MS MACRO ASSEMBLER	$99
87MACRO/DEBUG	$199
MICROSOFT FORTRAN V4	$299
RM FORTRAN	$399
LAHEY FORTRAN F77L	$477
MS or LATTICE C	CALL
STSC APL★PLUS/PC	$450
STSC STATGRAPHICS	$675
SPSS/PC+	$695
87SFL Scientific Functions	$250
87FFT	$200
OBJ → ASM	$200
PHOENIX PRODUCTS	CALL

287 Turbo™-10/12



287 Turbo runs the 80287 at 10 or 12 MHz in the IBM PC AT, compatibles and the new Compaq 386 with 100% software compatibility.

10 MHz **$450**

12 MHz **$550**

***PC Magazine* "Editor's Choice"**

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and installation instructions.

8087 5 MHz..... **$105**
For the IBM PC, XT and compatibles

8087-2 8 MHz..... **$154**
For Wang, AT&T, DeskPro, NEC, Leading Edge

80287-3 5 MHz..... **$179**
For the IBM PC AT and 286 compatibles

80287-6 6 MHz..... **$229**
For 8 MHz AT and compatibles

80287-8 8 MHz..... **$259**
For the 8 MHz 80286 accelerator cards and Compaq 386

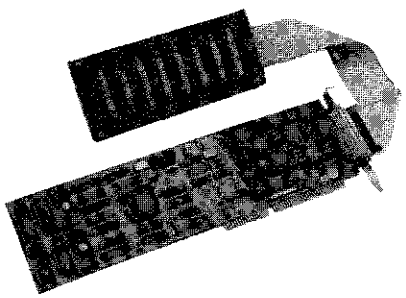
80287-10 10 MHz..... **$395**

PC-PAL™ Programmer..... **$395**

Call for great prices on V20, V30, 64K, 128K and 256K RAM

AT8™

Turns your AT into a high speed, multi-user Xenix business system!



8 port, intelligent serial controller with 3% response degradation. Includes 8 MHz 80186 with built in DMA **$1299**

MICROWAY SOFTWARE FOR LOTUS 1-2-3™

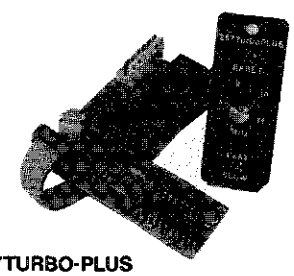
FASTBREAK™ employs the 8087 to increase the speed of Lotus 1-2-3™ Version 1A or 1A*. Users are reporting speed ups of between 3 and 36 to 1. When run with our NUMBER SMASHER accelerator card, recalculation speed ups of 10 to 30 are being reported..... **$79**

PowerDialer® Add-In for Lotus 1-2-3 Release 2. Automated telephone dialing from within 1-2-3. Adds least cost routing, automatic carrier selection and automated phone book worksheet. Builds customized dialing applications. Can be used with DesqView **$79**

HOTLINK™ adds easy linking of spreadsheets to Lotus 1-2-3 Version 1A... **$99**

287 TURBO-PLUS™ Speeds up your AT

Adjustable 80286 Clock 6-12 MHz
10 MHz 80287 Clock
Plus Full Hardware Reset..... **$149**
Optional 80286-10 ..... **$175**



287TURBO-PLUS
With 80287 10 MHz..... **$549**
With 80287 12 MHz..... **$629**

CALL (617) 746-7341 FOR OUR COMPLETE CATALOG

MicroWay
P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

You Can Talk To Us!

MicroWay Europe
32 High Street
Kingston-Upon-Thames
Surrey England KT1 1HL
Telephone: 01-541-5466

Number **104** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

more than 512K bytes of system tables. Intel provided a fast and smaller way to duplicate the 8086 addressing through a CPU mode. When this mode bit was set, the processor would compute addresses as the 8086 does.

But the so called "protected" mode was the one that allowed full addressing of 16 megabytes. In protected mode, pointers were 32 bits long: 16 bits for the segment index (or selector, as it's called) and 16 bits for the offset. All segment registers actually pointed to a system table that contained the real base of the segment, the size of the segment, and other information that allowed paging and protection on the segment level.

Protection was accomplished by restricting access to segments: for example, a data segment could be marked as read-only or a code segment could be marked as execute-only. Also added was the concept of a gate, a way to restrict entry points to a program and possibly change program access rights.

All of these features made the 80286 a very interesting processor to write an operating system for. It had, for example, all the primitives to write a MULTICS look-alike. MULTICS was a massive computer system created at MIT in the mid-1960s that combined paging and segmentation. Although it wasn't a commercially successful system, it was well documented, and many other computers adapted some of its ideas. The Unix system was an outgrowth of MULTICS although its design goals were much different.

One of the first operating systems to take advantage of the paging and segmentation hardware of the 80286 was XENIX, a Unix look-alike. But the reader should note that the largest object addressable was still 64K bytes large. This was not a big problem for Unix, which grew up on smaller machines, but it could be a problem for application programs that needed to use large amounts of data. This severe object size restriction was later addressed by the 80386 processor.

The 80286 did improve on the 8086 in the speed area, but except for the operating system instructions, instructions beyond the 80186's were not added. The 80287 math co-processor had a better interface to the processor and the program, but it was somewhat flawed by not being as fast as the 80286. Intel "solved" this problem with an on-chip clock divider that allowed it to work with the 80286 clock.

When the 80287 was made to run at higher speeds, several companies offered chip holders with separate clock crystals to run the chip at greater than the system clock speed. But the 80287 still did its calculations in the same way as the 8087 and had the same "bugs" as the 8087. These problems, too, would be addressed with the 80386's floating point processor.

The Intel 80386

By 1986, Compaq was shipping the 80386, the latest processor from Intel, in complete computer systems. Now, in the middle of 1987, numerous computer companies have announced available 80386 based systems or "turbo" boards that have 80386 chips. They run all existing programs for the IBM AT, and at least one turbo board, the In-Board by Intel, claims it will run OS/2 even though the operating system isn't released yet. This claim actually appeared in one In-Board ad, but as you can imagine, an ad that claimed it was compatible with vaporware (their word) didn't run for long! Clearly, the 80386 is the machine to follow. Through careful design, it can run all legal 80286 programs. And it is quite fast.

But the full power of the 80386 will not be realized until programs are converted to make use of added instructions and registers. Except for the segment registers, all registers in the 386 are now 32 bits wide. Effective address calculations use 32-bit offsets and the 16-bit segment selector. This means that a segment can be up to 4 gigabytes long and a program can have up to 16K objects this size.

Just ask Microsoft how many pounds the developer's package weighs.

Addressing is also easier with the 386, thanks to the two additional segment registers, the ability to use any 32-bit register as an index register, and the ability to scale index registers by the constants 1, 2, 4, and 8.

Various instructions have been added, and slow instructions have been speeded up. A special shifting circuit has been added to allow all shifts to occur in one clock cycle. There are a couple of instructions to shift paired 32-bit registers, which could make software floating point routines run at reasonable speeds.

There is also a new instruction to move a byte or word to a register and extend the sign of the source. Bit testing and setting as well as scanning for bits are now supported. And there is a set of instructions for setting a byte to zero or one depending on the condition flags. These come in handy when emitting code for languages that have boolean expressions.

Several features were added to the chip to make the operating system easier to write or debug. Believe me, this is a significantly better processor than the 80286. The 80387 floating point processor is also an improvement over the previous processors. Intel has fixed bugs and added instructions that speed up calculations of functions such as sine and arctangent.

Where Have You Been, IBM?

Here we are, at least a year after a complete 80386 product has been available, and IBM/Microsoft has announced its intention to market an operating system that will better use the 80286. They announced the new operating system simultaneously with the introduction of the Personal System/2 line of products, so it looked like the PS/2 would be required to run this new operating system confusingly called Operating System/2.

But this is not the case. Any AT-like machine will do. And the PS/2 model 30 won't run OS/2. All you seem to need is the 80286 and the AT BIOS ROMs. (Remember the Intel InBoard ad?) So this operating system is far more interested in supporting old hardware than in supporting the new (last year's) hardware. And it isn't even going to be available to users until next year!

So what's OS/2 all about?

It's big, and it's expensive. IBM is recommending at least 1.5 megabytes of RAM and 20 megabytes of disk space. OS/2 will run existing PC DOS programs, but only one at a time. It will eventually be very graphics-oriented like the Apple Macintosh, the Amiga, or the Atari ST, but the graphics interface won't be available on the first release. OS/2 should be able to support many different types of graphics cards and hardcopy devices. The list price for end users (not developers) is several times greater than PC DOS 3.2.

Almost every part of the system is designed to be replaced. Hackers could have a field day; all they need is the pile of documentation. And I mean a pile. If you don't believe me, just ask Microsoft how many pounds the developer's package weighs. And then ask them how much a developer's package costs, and ask yourself where a hacker is going to get that kind of money.

Programs will have to allocate memory as they need it. There are operating system services to allocate memory blocks larger than 64K bytes, but unless OS/2 gets changed to define a simple relationship between the the first segment used in a block and the others, it will take a call to the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

operating system to calculate addresses to inside objects larger than 64K bytes. This extra call will add significantly to the time it takes to access data. I know operating systems hardly ever manage objects that large, but applications programs do. (Are you reading this in Armonk?)

Unfortunately, the one part of the system that can*not* be replaced is the file system. This is the part of the system that restricts the maximum size of files and disk drives to 32 megabytes. Also, OS/2 will not support the new features of the 386 that make it so useful. For example, OS/2 will not preserve the 32-bit registers during task switches. Running in an environment where your registers may be destroyed at any time will make it hard to write reliable programs. But saving all the 80386 registers is not all that's needed to properly support the 386 processor. All, or at least most, of the operating system's calls ought to be able to use 32-bit pointers.

It is hard to imagine an operating system built in the late 1980s that would have such flaws. But then maybe it wasn't trying to be all things to all people. OS/2 offers a lot of ability to support graphical user interfaces, communications to mainframes, and tiny local databases. If I were a bit more paranoid, I would say they were trying to restrict the AT or PS/2 lines from competing with IBM mainframes. But this conclusion doesn't make total sense—mainframes typically have high I/O throughput, and the PS/2 line improved I/O throughput greatly with the MicroChannel, which helps alleviate one of the traditional weaknesses of PCs.

In 1987, it is hard to get excited about an undelivered operating system based on an already outmoded chip. I had hoped for more from Microsoft and IBM. Perhaps if this operating system had been available when the AT was shipped in 1984, I would have been more enthusiastic. But not now. And Intel is not standing still; they produce new processors every 2 years.

Producing a new operating system that supports a processor that is two generations old is not my idea of keeping up with technology. But then, closely following state of the art technology has never been IBM's forte: Their concern has always been solving the customer's problem. And in this case, their perception is that the OS/2 can do that. I'm obviously not convinced.

If your application can fit into this limited environment or you need some of the special features found only in OS/2, go ahead and develop for this operating system; otherwise, I'd suggest that you wait for something to support newer processors.

Reference

The *Microprocessor Handbook* by Joseph D. Greenfield (John Wiley & Sons, 1985) is an excellent reference for the history of microprocessors from the Intel 8080 through the 8086. Greenfield also covers the Zilog parts Z-80 and Z-8000 and Motorola parts 6800, 6809, and 68000. □

Christian Walker is a founder and one of the numerous vice presidents at True BASIC, Inc. He has been responsible for the IBM PC versions of the True BASIC language systems, which also run on the Apple Macintosh, the Amiga, and the Atari ST. He is currently porting True BASIC to an 80386 running in protected mode. Send constructive criticism to:

True BASIC, Inc.
39 S. Main Street
Hanover, NH 03755

MULTITASKING AND MULTIUSER PERFORMANCE FOR 27¢ PER DAY

With **OPERATING SYSTEM TOOLBOX,** the Software Construction set that allows you to build your own custom operating system.

With **PCNX,** Wendin's PC version of Bell Laboratories' incredible UNIX Operating System.

With **PCVMS,** A system similar to the VAX/VMS Mainframe Operating System that brings an elegant set of services to the PC.

FROM WENDIN . . . of course!

EACH PRODUCT IS PRICED AT ONLY $99.00, AND INCLUDES SOURCE CODE ON DISK.
If you average that cost out, that's only 27¢ per day for one year. THE REST OF YOUR LIFE IT'S FREE!

★★★ **MENTION WHERE YOU SAW THIS AD** ★★★
WHEN YOU ORDER AND RECEIVE FREE A HIGH QUALITY PEN AND PENCIL SET IN A DELUXE GIFT BOX COMPLIMENTS OF WENDIN.

ORDER HOTLINE (509) 624-8088

Wendin is a registered trademark of Wendin, Inc. PNX, PCVMS, and Operating System Toolbox, are trademarks of Wendin, Inc.

WENDIN®
BOX 3888
SPOKANE, WA 99220-3888

Working beyond the horizon to develop the operating systems of tomorrow
© Copyright 1987 Wendin, Inc. (509) 624-8088

Number **77** on the Reader Service Card

TURBO SOURCE SEARCH

"BBS-by-mail"
2 FREE DISKS then just $2.50 A DISK

TURBO SOURCE SEARCH (TSS) is the largest SIG library of Turbo Pascal public domain/shareware programs.

TSS is in fact a Bulletin Board System (BBS). We have the usual modem-connect BBS (617-545-9131), but also offer a BBS-by-mail service where members may get the files from the library on disks by mail for just $2.50.

TSS members pay a one time fee of $25 and receive a set of catalog disks plus credit for 2 FREE DISKS.

TSS has over 110 disks, 35Mb of compressed files, of software. These files have been placed into public access channels by their authors and collected by TSS so that you can avoid the costs of collecting them yourself.

And since TSS is turning ONE YEAR OLD we are offering a one price special of $250 for full membership PLUS the full library of disks updated through disk 150 which should be added to the library by mid-summer!!! (Current members: $2 per disk, 50 + disk orders.)

For a set of catalog disks send $5, or send $25 to become a full member of TSS, the "BBS-by-mail" to:

TURBO SOURCE SEARCH
P.O. BOX 876, SCITUATE, MA 02066, 617-545-6677
data/BBS: 617-545-9131 (anyone may call!)

VISA, MASTERCARD, COD $5 extra, non-usa orders add $10

Number **35** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

An Interview with Jack Davis, President of Metagraphics,

Or, I Never Meta Man I Didn't Like

by
Frank D. Greco

At a PC trade show in New York City about two years ago, an amazingly quick graphics demonstration caught my eye at the Tecmar booth. A self running demo displayed a series of overlapping windows that snapped open, displayed some graphics-oriented information, and closed at a speed surprising on an ordinary PC. Since I was already using a sluggish window-based 68K-powered Unix workstation, the amazing performance of this graphics software on the 8088-powered PC grabbed my attention. After some maneuvering with the Tecmar sales representative, I found out that the software was derived from a small company in California called Metagraphics Software Corporation.

Their product, MetaWINDOW, also attracted the attention of *PC Tech Journal*. In fact, it captured their Product of the Month slot in July 1985. Now with PC graphics and windowing products commanding much more attention than text-mode only systems, it makes sense for all software developers to investigate high performance graphics tools for incorporation in their applications. Personally, I was very impressed with Metagraphics' MetaWINDOW.

I recently had an opportunity to chat with Jack Davis, president of the Metagraphics Software Corporation. The following is a result of that conversation.

FG: Jack, could you briefly describe MetaWINDOW?

JD: MetaWINDOW is a high performance graphics toolkit that provides PC program developers with a powerful set of basic drawing functions plus an expanded set of advanced utilities for developing multi-window desktop applications. The system is based on "object-oriented" graphic display concepts such as those pioneered at the Xerox Palo Alto Research Center (Xerox PARC) and is very similar to the graphics systems used on other modern windowing systems such as the Xerox Star and Apple Macintosh and the Sun Microsystems and Apollo workstations.

MetaWINDOW fits in the gap left between low level graphics primitive libraries such as GKS, VDI, and CGI from GSS or Halo from Media Cybernetics on the one hand and fully pre-packaged window managers such as Microsoft's Windows or Digital Research's GEM.

MetaWINDOW is not a prepackaged window management system like Microsoft's Windows, Digital Research's GEM, or Sun's SunView; rather, it is a toolkit that can be used for developing window management systems.

Probably a closer analogy for MetaWINDOW would be to Apple's QuickDraw system that provides the graphics foundation for the Macintosh.

FG: What advantages does MetaWINDOW offer over the Windows and GEM toolkits?

JD: Since MetaWINDOW is used as a toolkit for developing window managers, the application developer has full control over the design and appearance of the user interface. Also, since the designer can pick and choose only the functions he needs to use, the size and performance of the window manager can be optimized to a much greater extent than with prepackaged window managers.

This is especially important for large applications. MetaWINDOW's minimum memory requirement is about 20K bytes although about 45K is typical for most applications. A small portion of code still needs to be written for designing the window manager, but this is still far less than the 256K minimum, 512K+ recommended, for Microsoft Windows or GEM. Our full bitmap font/icon editor program, for example, uses a Mac-like window manager and was written in about 6 pages of C code; it is only about 150K bytes in size.

And MetaWINDOW is much more highly optimized. Our benchmark comparisons indicate about a 2x performance advantage in using MetaWINDOW over Microsoft Windows or GEM. MetaWINDOW's performance actually comes as a double bonus—performance by the elimination of device drivers! MetaWINDOW's "object-oriented" design focuses on graphics functionality rather than specific manufacturer hardware products. This technique concentrates on optimizing the functionality of the package as a whole, rather then writing and supporting multitudes of separate device drivers for each manufacturer's hardware. Device definitions within MetaWINDOW are simply stored as short table entries that can be referenced dynamically when the program executes. MetaWINDOW even includes a special procedure, QueryGrafix( ), which automatically determines the hardware configuration installed on the system it's running on. The result eliminates complex configuration programs or the myriad separate installable device drivers—the program simply executes and comes up running—all this with the added bonus of higher performance and smaller code requirements.

MetaWINDOW also includes many more graphics functions and supports a wider range of graphics peripherals than either Microsoft Windows or GEM. MetaWINDOW's open architecture also makes it easy for developers to add support for new or custom display devices not directly included in the system.

FG: Has any of your customers taken your product and built a window manager layered on top?

JD: Our customers have designed a number of different styles of window managers for their own use. As part of our customer-contributed library we have a basic Macintosh-like window manager. With the extended windowing utilities provided in MetaWINDOW, window managers are typically quite simple to implement. Using the application tools in MetaWINDOW, a programmer can write his/her own style of window manager, tiled or overlapping. MetaWINDOW provides the necessary bitmap and port data structures, but the programmer has full control over how the user interface is designed.

FG: Does MetaWINDOW require the power of an IBM PC AT (or any clone using an 80286 or 80386)?

JD: Since MetaWINDOW concentrates by function on the graphics side of the window manager, we've highly optimized this portion of code within MetaWINDOW. As I've said,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

our benchmarks have demonstrated an average 2x performance advantage over the equivalent functions in Windows or GEM. We've had developers switch to MetaWINDOW specifically because it makes their applications viable on standard PC and XT systems. Even for 286-based AT systems, MetaWINDOW can substantially increase the capacities an application program can handle.

FG: Of all the Xerox graphics/windowing systems (Smalltalk, Cedar, Viewers, Tajo) which are you most related to?

JD: MetaWINDOW is probably most similar to the Smalltalk-style of object-oriented graphics.

FG: Could you describe the two versions of MetaWINDOW? As far as I can tell, you have a DOS-resident version and a library (non-resident) version.

JD: MetaWINDOW is available in two basic forms: standard MetaWINDOW and MetaWINDOW/PLUS.

The basic MetaWINDOW package provides the graphics routines in the form of a DOS resident driver—a program that is executed first and remains resident as an extension of the operating system. The interface libraries provided for the different languages and memory models then access the driver using an interrupt vector in low memory. This driver form of MetaWINDOW provides advantages for languages that compile for "small" memory model configurations. Since the graphics driver is external to the application, only about 4K bytes of the available 64K code space is needed for the MetaWINDOW interface library. The driver form of the package also offers an advantage in that we do not require any additional licensing to redistribute the MetaWINDOW graphics driver with a developed application.

MetaWINDOW/PLUS provides the graphics routines in the form of directly linked object module libraries (multiple libraries are provided for the different languages and memory model configurations). MetaWINDOW/PLUS provides additional memory and performance options and is oriented for OEM application use. Since the application program links directly to the graphics procedures, it links in only the routines actually used by the application. They may be as small as 20K bytes, but are more typically 40K to 50K bytes—versus the 75K bytes for the MetaWINDOW DOS resident driver that loads the entire MetaWINDOW system. Also, since the application is typically using only a subset of the entire graphics system, the MetaWINDOW/PLUS libraries are conditionally assembled for performance optimization first, memory optimization secondly. This differs from the standard MetaWINDOW package, which assembles the resident graphics driver for memory optimization first, performance secondly, to keep the driver as small as possible.

A third product, TurboWINDOW, is also available, which provides a MetaWINDOW interface specifically for use with Turbo Pascal. [Libraries for Borland's Turbo C and Microsoft's Quick C are currently being written.——FG]

FG: Which interrupt vector does the DOS resident version use?

JD: The driver defaults to use interrupt 7C (hex). This can be changed by the user with a command line switch specified when the driver is installed. In addition, there is a conversion utility that can be used on the MetaWINDOW library. The conversion utility links to the application, or it can be used on prelinked .EXE/.COM file programs.

FG: Does the programmer access your DOS-resident version in the same familiar fashion as any other DOS interrupt?

JD: The programmer accesses the MetaWINDOW graphics routines as he would any other Pascal, C, or Fortran procedure. The interface libraries provided with the package take care of redirecting to the graphics driver automatically.

Technically, the calling sequence to the graphics driver differs from the DOS interrupt 21 technique, which fixes the number and form of parameters passed. The driver call is quite simple and can be broken down into three steps:

1. First the parameters are pushed onto the stack.
2. Register BX is loaded with a function identifier indicating the MetaWINDOW procedure being called.
3. An indirect call is issued through the interrupt vector in low memory to access the graphics driver.

FG: Can or will MetaWINDOW use any of the extended memory boards (above 640K bytes) out there, or are you leaving that issue to the responsibility of the developers using MetaWINDOW?

JD: MetaWINDOW currently supports the Intel Extended Memory Specification such as available in the Intel AboveBoard and Tall Trees jRAM products.

One of the more exciting uses of extended memory is in combination with Tall Tree's jRAM card and jLaser printer interface. This combination provides 2 megabytes of on-board memory and can be used to create full page text and graphics images for direct laser printer output at 300 dots per inch!

FG: Does MetaWINDOW break any of the DOS "rules"—for example, writing directly to video memory? And if it does, do you foresee any problems with the multi-tasking, protected-mode version of Microsoft's MS DOS, that is, OS/2?

JD: The only "rule" that MetaWINDOW does break is directly accessing video memory—this, in part, is how we gain our enhanced drawing speed. Unless Microsoft purposely chooses to shoot themselves in the foot, we expect they will maintain a system mechanism allowing access to specific memory areas. This is required not only for efficiently supporting display adaptors, but also for many other hardware products.

FG: Does MetaWINDOW support color?

JD: MetaWINDOW provides full color support for most popular display adaptors, including the IBM Enhanced Graphics Adaptor (640 x 350 16-color), the AT&T Display Enhancement Board (640 x 400 16-color), and the Number Nine Revolution (512 x 512 256-color).

FG: Does MetaWINDOW contain any sort of higher level memory management scheme for bitmaps?

JD: MetaWINDOW itself provides no memory management utilities, but rather uses the

Our benchmark comparisons indicate about a 2x performance advantage in using MetaWINDOW over Microsoft Windows or GEM.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DeSmet C
SUPPORTS
ROMABLE CODE

DeSmet C ***$109**

Data & Stack in RAM
Code executes out of ROM
Full K&R + V7 extensions
Inline asm
Assembler, linker, librarian
Full screen editor (SEE[tm]),
8087 & S/W floating point

DeSmet C with D88 Debugger **$159**

Set Breakpoints by line number or function name
Examine Global/Local variables by name
Show C source while debugging
Both D88 and User displays

Large Memory DeSmet C w/ D88 **$209**

32-Bit Pointers (Full Megabyte Addressing)
Fast Access of Static Data
Includes the standard compiler features mentioned above.

*Both D88 & Large Memory available as options at $50 each.
We also have a C compiler for the Mac. $150. Call for details.

— plus —

Graphics **$35**
BASICA-like (+src)

Hacker **$25**
Source for start-up, RAM.COM & more

Make **$50**
Full UNIX-level

Tools (+source) **$35**

XARRAY **$39**
Large Arrays (+source)

C WARE
CORPORATION
P.O. Box C
SUNNYVALE, CA 94087 USA
(408) 720-9696 TELEX: 358185
Street Address: 505 W. Olive, #767
VISA, MC & AMEX accepted

Number **38** on the Reader Service Card

memory management facilities of the compiler it's working with. Allocation of local memory bitmap space, for example, is performed by the application, and a pointer to the allocated space is then passed to MetaWINDOW.

Since the current DOS does not provide a complete built-in memory management system, we must be compatible with—and not interfere with—the various memory management systems used by the different compilers. This doesn't really affect MetaWINDOW right now, but currently restricts the implementation of higher level window management functions needing memory management to one particular compiler. IBM's next DOS reportedly includes memory management support, at which time we'll be able to provide a more complete set of language independent, high level windowing functions.

FG: Which languages can MetaWINDOW be interfaced with?

JD: MetaWINDOW includes direct support for most popular C, Pascal, and Fortran compilers including Lattice C, Microsoft C, IBM C, Computer Innovations C, MetaWare C, IBM Pascal, Turbo Pascal, MetaWare Pascal, Microsoft Pascal, IBM Fortran, Lahey Fortran, Microsoft Fortran, and Assembler.

Where languages support multiple memory models, MetaWINDOW includes separate interface libraries for each memory model.

FG: For those languages and vendors' products not directly supported by MetaWINDOW—Aztec C, Wizard C, Modula 2, dBASE II/III, interpreted MBASIC)—I presume that an experienced programmer who has sufficient information regarding the interrupt vector and entry points can write his or her own interface code using an assembler or some generalized DOS interface.

JD: The MetaWINDOW manuals contain the information needed to access the graphics driver from other languages. For compilers that operate with the standard DOS linker, we've even automated the interface process with a pre-programmed set of interface utilities called MetaBUILD. These utilities include a library of assembly language source files defining the driver calling sequence for each of the MetaWINDOW routines, plus a common "include" file that defines the compiler calling convention through an assembly language macro definition. Once the "include" file macro has been edited for the new interface, a batch file provided assembles the procedure interfaces and builds the new interface library. The whole process can be performed quite quickly—about twenty minutes of reading, ten minutes of editing, sixty minutes of batch assembling, and voila!—you have a new MetaWINDOW interface library.

This technique does have one limitation in that it works only for compilers that use the standard DOS linker. Also, it requires some assembly language experience to define the compiler interface macro; however, we do provide working examples of the interface macros for the standard compilers we support. The MetaBUILD system is provided under a no-charge license—our only request is that we be provided a copy of the completed interface macros and library, which we can then make available to other users as part of our Customer Contributed Library.

FG: How portable is MetaWINDOW? Is it strictly an 80x86 product, or are you considering porting it to another microprocessor family?

JD: We do our prototype development using C, and then code to assembler for memory and performance optimization and language portability. The C version is highly portable though we usually need to code to assembly language for maximum performance and memory optimization in a package of this type. Other than developing support for the IBM RT, we will be concentrating our efforts on expanding the capabilities of MetaWINDOW with supplemental products for the IBM PC through the end of the year.

MetaWINDOW itself is highly system independent. Other than for file I/O access for loading font files, there are no DOS dependencies at this time. Although there has been a lot of discussion of Unix and XENIX on the PC, we really haven't seen that market take off. Right now we're in a watch and listen posture regarding new operating systems—watching the market and listening to our customers. We expect that IBM's multi-tasking operating system (OS/2) will be our next major PC based operating system upgrade.

FG: Does MetaWINDOW support some sort of VDI (Virtual Device Interface)?

JD: MetaWINDOW is quite highly device independent but makes no claims to be a GKS/VDI compatible interface. Looking back, we can see that graphics software technology has really evolved from two basic origins: the first from "stroked" graphics roots originally promoted by companies such as Tektronix and now seen as GKS/VDI implementations; the second from raster graphics roots, such as originally pioneered by Xerox PARC, and now seen in "object-oriented" graphics systems on the Apple Macintosh, Xerox Star, Sun Microsystems, Apollo, and MetaWINDOW systems. We've had several users build short GKS/VDI interfaces to MetaWINDOW for compatibility with some of their earlier programs, but the basic "stroke" oriented design of GKS tends to limit its functionality and slow its performance on raster systems too much to make it of much general use.

MetaWINDOW does provide its own virtual device capabilities, which are actually much more portable and efficient than GKS/VDI.

FG: What input and output devices do you currently support?

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

JD: MetaWINDOW currently supports over 30 of the most popular graphics adaptors for display output: Microsoft, Mouse Systems, and Logitek mice for graphics input; and IBM/Epson dot matrix printers plus the Tall Trees jLaser laser printer interface for graphics hardcopy.

FG: Could you give us a little history of your company?

JD: There were five principals involved in starting Metagraphics. Most of us met through contacts at Xerox PARC and became quite familiar with the object-oriented graphics technology developed for the Xerox Alto and follow-on Star systems. I personally started with a company called Versatec back in the early '70s.

FG: Which version of your product has attracted the most commercial attention?

JD: More recently, the special Turbo Pascal version of MetaWINDOW, TurboWINDOW, is getting attention. Many colleges and universities using Turbo Pascal to teach programming have now begun introductory computer graphics courses using TurboWINDOW—MIT, for example.

FG: Considering that only about 7% of all PCs out there have mice connected to them, do you think corporate America (or even international businesses) will eventually accept mice, icons, and graphics windows?

JD: The rate at which mice are accepted will be determined by the availability of software that makes effective use of them. Right now there are relatively few packages on the PC supporting mice as an integral part of their operation. In time, especially as the software market continues to become more competitive, I believe we'll see more vendors adding mouse support to simplify and enhance the friendliness of their products.

FG: What type of licensing agreements, if any, do you have with developers who are considering incorporating your MetaWINDOW code into their own?

JD: With our standard MetaWINDOW and TurboWINDOW products, the license agreements allow program developers to freely incorporate and redistribute the MetaWINDOW graphics driver with their applications without any additional licensing or royalty requirements. I believe we're the first and only major graphics system to take a non-royalty stand on its basic products.

For our MetaWINDOW/PLUS package, intended for larger developers and OEMs requiring added optimizations, support, and maintenance, a separate paid license agreement is required when MetaWINDOW/PLUS is distributed within a product for resale. Even this license agreement pertains only to using MetaWINDOW/PLUS in products for resale. No additional licenses or royalties, even for MetaWINDOW/PLUS, are needed for educational, inter-company, or other non-resale uses.

We've tried to be as flexible as possible in our licensing arrangements, providing virtually unrestricted use with our basic MetaWINDOW and TurboWINDOW systems while providing MetaWINDOW/PLUS for larger developers and OEMs who need its additional optimization, support, and maintenance features.

FG: What does the future hold for MetaWINDOW and Metagraphics?

JD: We are currently completing a full spectrum of graphics related products for software developers and OEMs:

- TurboWINDOW - MetaWINDOW specifically for use with Turbo Pascal
- MetaWINDOW - Standard MetaWINDOW (multi-language/multi-device)
- MetaWINDOW/PLUS - MetaWINDOW optimized for OEM application use
- MetaFONTS - MetaWINDOW bitmap font/icon editor and basic font development tools
- MetaFONTS/PLUS - MetaWINDOW outline font editor for creating and editing PostScript compatible filled-outline fonts
- MetaSCRIPT - PostScript output capability for MetaWINDOW
- MetaPLOT - Plotter output support for MetaWINDOW

With graphics technology still rapidly evolving, we look forward to expanding the capabilities of MetaWINDOW over the growing range of higher performance, higher resolution, and lower cost graphics peripherals. PC graphics is really just in its infancy, and with the next generation, things should really get exciting! □

Programmer's Journal *would like to thank Mr. Jack R. Davis of Metagraphics Software Corporation for his assistance and cooperation in this interview.*

Metagraphics Software Corporation
4575 Scotts Valley Drive
P.O. Box 66779
Scotts Valley, CA 95066
(408) 438-1550

Frank D. Greco is an independent consultant who specializes in highly usable computer systems. Readers can contact Frank at the following address:

Frank D. Greco
c/o Crossroads Technologies, Inc.
P.O. Box 530
Fanwood, NJ 07023
(201) 754-7820

DAN BRICKLIN'S DEMO PROGRAM ONLY $74.95

Read what they're saying about this popular program for prototyping and demo-making:

"A winner right out of the starting gate. After you use DEMO once, you'll wonder how you got along without it."
—PC Magazine

"Everybody who writes software, either commercially or for in-house applications, should immediately order a copy. Period. No exceptions."
—Soft·letter

Product of the Month
—PC Tech Journal

Thousands of developers and most of the largest and best known software companies are using this program. You can, too. Act now!

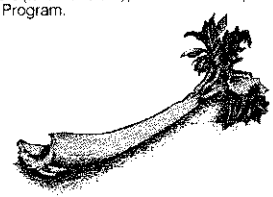
NEW TUTORIAL! JUST $49.95

The perfect companion to the Demo Program. The Tutorial helps you learn the ins and outs of its basic and advanced features. Complete with a 96 page manual containing step-by-step instructions, diskette, and function key template.

ORDER NOW!
1-800-CALL-800 x8088

VISA MasterCard AMERICAN EXPRESS

Use 800-number for orders only.
Questions, special shipping, etc., call **617-332-2240.**
No Purchase Orders. Massachusetts residents add 5% sales tax. Outside of the U.S.A., add $15.00.
Requires 256K IBM PC/Compatible, DOS 2.0 or later. Supports Monochrome, Color Graphics, and EGA Adapters (text mode only). The Tutorial requires the Demo Program.



SOFTWARE GARDEN, INC.
Dept. J
P.O. Box 373, Newton Highlands, MA 02161

Number **108** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Making Computer Tools Pay for Themselves

by
Paul Barkley

Something's Wrong with All This Neat Stuff

I sometimes think it's a wonder that anything at all is accomplished with the use of computers. We're all aware of the marvelous advances in hardware and software that seem to arrive regularly (albeit not necessarily on schedule). Compare any piece of hardware or software with the same object five years ago, and you'll see what I mean. What word processor were you using five years ago, and how does it compare with what you're using today? The old one, which you no doubt were very fond of, seems pretty primitive now.

Yep, we've come a long way in a few short years, but I'm starting to get uneasy about the real progress that we've made. On every objective comparison of hardware and software, we're forging ahead, but what about system effectiveness?

"What the hell is system effectiveness?" you ask.

To me it means, "How much work, and of what quality, can we get done with this stuff?" Smartalecks started calling the Macintosh the "Unproductivity Machine" because it was so much fun to play with that no one got a lot of work done—at least initially. Desktop publishing (DTP) systems offer the same trap. It's easy to get caught up in the wonder and power of the system and wind up not getting much accomplished.

In many cases we fail to realize the extent of the learning curve. It doesn't take long to become reasonably proficient with a new word processor. It's a natural extension of the other word processors you've learned to use. It won't "feel right" for some time, but you'll still be able to get a moderate amount of work done.

Unfortunately, it may take many months to become an expert in the product, and in that time you won't be working at anything near peak efficiency. (We'll assume here that the new word processor offers some substantial improvements over your old software and that you weren't forced into some "devo" product for political reasons—in that case you'll never be fully productive. If you change jobs, you may well be forced to learn a new word processor that offers no improvements over your old one simply because you need to be compatible with your new office-mates.)

Learning a new word processor is onerous but not a total disaster. We could probably make the same statement about spreadsheets. Once you've learned one spreadsheet, learning others is not that difficult.

But what about something more radical, such as desktop publishing? That's where productivity disappears down the same hole Alice fell into. (Somewhere the White Knight and the Red Queen must be cranking out incredible quantities of memos and position papers, up to their necks in all that missing productivity.) DTP is radically different from what you've done before, and your learning curve is going to be much flatter than you thought. What you need in DTP, if you're like me, is some graphic design and layout skills.

Sure, we're seeing style books now that can make almost anyone look good. Just be careful not to get too experimental until you learn something about the graphic designer's trade. We've all seen grotesque examples of trivial work produced in 15 typefaces on the same page, with two typos and a grammatical error thrown in for good measure.

Whenever Possible, Change Everything Around Until It No Longer Works

We often make things a lot more complicated than we need to because we have all these nifty technological tools we don't know how to use properly and don't know how to integrate with our other tools and our personal experience base. By the time we figure out how to make these new tools work, we're too bored with them to get any work done. Also, until we actually try the products ourselves, we have no way of knowing if they'll really do us any good—we're just guessing along with the computer salespersons.

I remember when I bought my laser printer and my new word processor and the font cartridge from a third company and tried to turn out new and improved documentation for my custom applications. I eventually got there, and today I'm pleased with the results, but I didn't plan on spending a month getting things in order and making them work smoothly.

Now, after a year of producing nice documentation, I'm no longer satisfied. I still cut and paste screen dumps for the documentation, and I use a color copier to reduce the screens and lay them into the final documentation in a contrasting color. Why? Because I don't have enough fonts or sizes, and I don't have enough control over graphics to make the screen drops visually interesting. I can't easily print reverses, showing input areas; I can't print the IBM extended graphics characters; and I can't properly box the screen dumps so that they look like little screens and not just part of the documentation text.

Okay, so my stuff looks a lot better than what someone might turn out with a (shudder) typewriter, and it looks better than a word processed document printed on a letter quality printer, but the problem is that I know I can do better, and I know just the tools to accomplish this task.

Let's see, first I need a Terminate and Stay Resident (TSR) program that properly captures screens with their reverses. Then I need a smart word processor or DTP program that can print the screen dumps and make them look good. I really ought to have the DTP system because of course I'll want to scale the screens down. Then I need a smarter laser that can handle more

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

fonts and sizes because getting the correct size for the screens, in relation to the fonts used for the explanatory text, might be tricky. I also want to box the screens, and that calls for some graphics ability in the word processor or DTP package and also in the laser.

Oh yeah, I need another computer to run the application while I write the documentation on the first computer, or I need some sort of memory resident software that will let me switch freely between the application and the word processor used to document it. Actually, in this case one job can be put to sleep while I run the other, so it's not as complicated as a true multi-tasking system.

So what's wrong with all this? Obviously, the system is getting out of hand. Although all the tools I discussed exist today, and in fact are not that unusual, we're looking at a major change. How long do you suppose it will take to get the new system set up? A week? Two weeks? Suppose something doesn't work right? (Does this sound to you like loathsome whining in the face of uncertainty, or does it sound like I'm voicing strong practical concerns?)

There's a tremendous learning problem here, the same sort of effectiveness trap that I alluded to earlier. The only difference is that now I know I'd better not start fooling around with this stuff until I can afford to set everything aside for a week or two. The *real* cost of the technology seems more apparent to me now than it used to.

This whole computer industry is like a giant dispersed factory run amuck. What would you say if you read that the bigwigs at General Motors just opened a new experimental factory where they build only two or three cars at a time and then tear down the assembly lines and set them up with new equipment? They keep getting more sophisticated robots and tougher materials faster than they can integrate this stuff into their lines. Since the technology is there, they build a couple cars one day, shut down for a month to retool, and then build a couple more cars. The real goal of the engineers is to build factory *lines*, not cars! Doesn't this sound a little strange and rather, uh, unprofitable?

GM might be able to fund one factory to play with because they're a Really Big Company, but ultimately they'd have to pick a configuration and then use it in a real factory to build tens of thousands of cars, not just one or two. They certainly couldn't run all their factories this way.

Computer installations are the same. I went to a lot of trouble to get my laser working the way I wanted it to, and then I cranked out work for only a year or so. Now I want to tear it down and start over. How efficient would that be? Is this a smart move? Have the tools paid for themselves yet?

You know, in your heart of hearts, that you're just the same as I am, and so are many of your clients. The technology is marvelous. It's sexy, as the marketing reps say. It turns you on. Technology, in its own way, is as addictive as designer drugs although it damages your physiology only maybe half as much.

You want to fool around with new hardware and software because it's so damn interesting, don't you? Remember what killed the cat?

tary B- school science. If it won't pay for itself (and then some), we don't need it. This isn't supposed to be a hobby.

So how do we make the calculation? Here's a sample. Let's say I upgrade my system to handle my document preparation. I add up all the costs, and I find that it will be cheaper, and much more sensible, to junk my present equipment rather than try to patch the processor and laser and monitor with strange add-on boards and all sorts of nonsense. Some of the old system is five years old anyway, and that stuff belongs in a museum. Okay, so all the pieces come to, say, $10,000 including the resale value of the old hardware and obsolete software.

How long will it take, in direct time, to install it, configure it, and learn to use it properly? Perhaps a day to install it and a week of heavy use to achieve a minimal proficiency. Given the radical change in equipment, the actual time will be longer than that—the keyboard will be different and cause eighty gazillion typos that will have to be fixed; the mouse will cause me no end of motor skill confusion; the monitor will need to be adjusted for brightness, contrast, height, tilt, and so forth. The laser won't work right because I'll have the wrong cable even though I saw it work at the dealer's with what looked like that cable. You get the picture. The more you change, the worse it gets. Remember the principle of scientific experimentation? Change only one variable at a time and observe the results. Hah! Change everything, and you won't have the slightest idea what's gone wrong.

It may take many months before you become an expert in the product, and in that time you won't be working at anything near peak efficiency.

(Can it be that as we grow older we grow ever so slightly wiser?)

Remember when I said it's a wonder that we get anything at all accomplished with computers? It seems that we're always setting up new things, and trying new combinations, and striving, gloriously, for our own versions of a brave new world. Just one more gadget, just that smallest gizmo will surely make our lives easier.

The Problem Revealed

"Ah ha," you cry triumphantly. "That's the whole problem! 'Far too much fiddling about,' as Uncle Ernie would say." I ask myself why you're quoting fictional characters from old rock operas and press on.

Getting Re-tooling Under Control

First, we've got to recognize the problem. It's a little like an alcoholic's admitting that he or she has more than a very slight problem. We can afford to upgrade on only a strictly controlled schedule. If a new piece of hardware or software costs X amount of dollars, we must use it to save X amount of dollars before we discard it. That's elemen-

Okay, Okay. So an entire person-month will be wasted by the time everything is truly working properly and the style sheets are set the way I want them. Of course there's then the matter of converting files, but we won't go into that. A smart new system should make that *almost* painless, but of course it will mangle at least something in translation.

Add up the value of your time. Ask yourself if you *ever* installed something with which you were not familiar in *less* than the time that you estimated the job would take. Sure you did.

Let's be optimistic and add only $4,000

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STOPWATCH

MEASURE EXECUTION TIME TO THE MICRO-SECOND

■ ***STOPWATCH™ . . . It Measures Program Time with Microsecond Resolution.*** The new STOPWATCH Program measures total and component times of programs executed on IBM PCs, XTs, ATs, and compatible units to within ± 5 microseconds.

■ ***STOPWATCH . . . It's Easy to Use.*** With STOPWATCH there's no need to modify your program for testing. Simply key ⟨ctl⟩-M instead of ⟨enter⟩ anytime you want to measure execution time to the next display output. (⟨ctl⟩-M and ⟨enter⟩ are equivalent *until* you install STOPWATCH.)

■ ***STOPWATCH . . . Use It To Time New Programs.*** Use STOPWATCH with DEBUG, SYMDEB or CODEVIEW to measure execution time from "GO" to Breakpoint. And, STOPWATCH can be used to time Lotus 1-2-3 Macros, dBASE III Procedures, BASIC Programs, or any object program that you have developed.

■ ***STOPWATCH . . . Use It To Evaluate Your Present System.*** Add STOPWATCH to your present system and measure component times of a program to quickly determine whether a new processor, or faster disk, is the key to improving response time for your application.

■ ***STOPWATCH . . . Unparalleled Accuracy At An Affordable $59.95.*** Included are demo looping programs to demonstrate STOPWATCH's timing ability, the STOPWATCH Program, and a user's manual with easy-to-follow instructions to get you started quickly. Now . . . Microsecond resolution timing — only $59.95! Put your program to the test today with STOPWATCH!

Now Microsecond Resolution Timing Only $59.95 +3.50 S&H + 6% NJ Tax. MC/VISA accepted

US 800-345-0167
NJ 201-228-7623

CRTS
CUSTOM REAL-TIME SOFTWARE

P.O. Box 1106, West Caldwell, NJ 07007

IBM is a registered trademark of International Business Machines Corp.
CodeView is a trademark of Microsoft Corp.
Lotus and 1-2-3 are registered trademarks of Lotus Development Corp.
dBase III is a trademark of Ashton-Tate
™STOPWATCH is a Trademark of Custom Real-Time Software Inc.

Number **124** on the Reader Service Card

in lost time to the $10,000 hardware and software price tag. That means we have to justify $14,000 in terms of savings.

Direct savings are straightforward. Let's assume that as part of the upgrade, the processor went from an 8086 to an 80286 or from an 80286 to an 80386 (well, maybe not an 80386 system like this for $10,000, but you get the idea). If you spend 30 percent of your time writing code, say 600 hours per year, you might be able to knock as much as 10 to 20% off that time by shortening the compile times, depending on your coding style. Call it 120 hours a year. Wow!

The new documentation effort will proceed more quickly because of the pseudo multi-tasking and because you eliminated cut and paste entirely. If you spend 20 percent of your time writing documentation (400 hours per year), knock off another 20 percent, or 80 hours per year. Oh boy!

Perhaps these are the only direct costs that we can identify in our example. In that case we have a total of 200 hours per year in time saved. Let's make an extraordinary leap of faith and assume that we can bill all of this saved time at our usual $60 per hour. That means we can earn an additional $12,000 per year with the time saved.

Since the system cost us about $14,000 and we can earn an additional $12,000 per year (which handily works out to $1,000 per month), the payback period is about 14 months. This means that we must leave the entire system alone for the next 14 months in order to get our money back, and we must leave it alone longer than that in order to start making any money from our investment. That's the key!

This was a simplistic example. In reality, the payback figures I used were too optimistic. It will take longer to master the new system. We won't be able to bill all of the hours saved but will wind up squandering some of them on activities like reading more junk mail and planning how to spend our fortune whenever we make it. In addition, there's an absolute certainty that we won't be able to leave the system alone for the next 14 months without at least adding some new piece of software.

Offsetting all this are the intangible benefits from producing higher quality work. What's it really worth to us to have compiles go fast enough that we don't lose our train of thought and wander off to the cookie jar? What's it worth to our business to produce stunning documentation? (Sad to say, no client in the history of computer consulting has ever read more of the documentation than the page that has the phone number to call with questions.)

One of my constant themes is the need for consultants to make themselves as productive as they can. In earlier columns I've said that effectiveness is more important than simple efficiency. If you can produce classic code and dynamite documentation in a cost effective way and can take care of marketing, your business will prosper. You can't measure the value of the improved tool.

The point, though, is that you can't afford to be constantly messing with your tools anymore than GM can afford to keep messing with the factory lines. You must use the tools for *production*, and when you tool up, you must make those tools pay their own way by producing a lot of product.

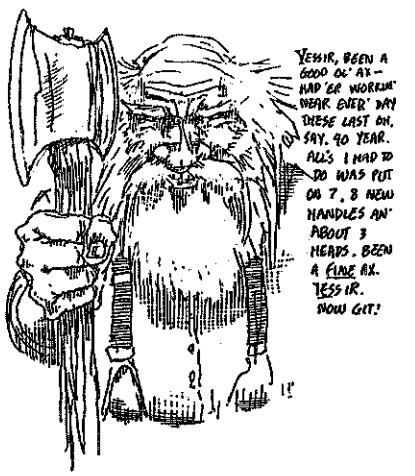
This trap affects us in many ways. How many times have you started to build a software tool and then either abandoned it because you got busy or spent 29 percent of the next year "perfecting it?"

I've said several times that if you have slack time, you should use it to develop your skills and to build products for your own use—consultant, automate thyself. The trick is to get the product to a stage where it's useful and then leave it alone. Actually, use it for a year and make note of all the improvements you'd like to make. Then, set aside a large block of time and fix the damn thing. When it's fixed, start using it again.

In either event, whether you buy tools or build them yourself, keep them in production long enough to more than recover the full cost of implementing them. Make the same decisions for yourself that you try to emphasize to your clients: Every new tool must be (dramatic pause) cost effective. □

Write with your consulting ideas and experiences to

Paul Barkley
The Barbican Company
6845 Elm Street
Penthouse One
McLean, VA 22101


YESSIR, BEEN A GOOD OL' AX— HAD 'ER WORKIN' NEAR EVER' DAY THESE LAST OH, SAY, 90 YEAR. ALL'S I HAD TO DO WAS PUT ON 7, 8 NEW HANDLES AN' ABOUT 3 HEADS. BEEN A FINE AX. YESSIR. NOW GIT!

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

You'll be striking into rich business territory when you exhibit at the 10th Northeast Computer Faire and the 13th West Coast Computer Faire.

Both shows bring you the audience niche you need, matching you up with the vertical market buyers and decision-makers, the computer literate trendsetters and businesspeople who've made the Northeast Computer Faire the longest running trade show in New England and the West Coast Computer Faire the first, biggest (43,500 attendees pored over the exhibits last year), and longest-running end-user event on the West Coast.

Add to this litany of superlatives the fact that Computer Faires bring you more leads, more new customers and contacts, higher sales and profits, strengthened business ties, and better name recognition, and you can see why scores of exhibitors are signing up for prime exhibit space at both shows at a scorching pace.

Light up your sales with the perfect matches. Call (617) 449-6600, ext. 4014 for information on exhibiting at the West Coast Computer Faire and Northeast Computer Faire. Or fill out and send in the coupon below to Computer Faires, 300 First Avenue, Needham, MA 02194. Strike while the iron's hot!

THE 10TH NORTHEAST COMPUTER FAIRE
World Trade Center, Boston, MA
October 15–17, 1987

THE 13TH WEST COAST COMPUTER FAIRE
Moscone Center, San Francisco, CA
April 7–10, 1988

I Want the Perfect Matches!

☐ Send me complete information on exhibiting, including brochure, contracts and floorplan for
☐ Northeast Computer Faire
☐ West Coast Computer Faire
☐ Have a Sales Representative call me now!

Name_____
Title_____
Company_____
Address_____
City_____ State____ Zip_____
Phone___ (_____)

Send to: *Computer Faires, Marketing Dept.*
300 First Avenue, Needham, MA 02194. PJ S/O87

Computer Faires.
The Perfect Matches.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Case of the Invisible CapsLock

by
Tom Swan

OKAY, MR. SWANNO, WE'RE ALL TIRED & WOULD LIKE TO GO HOME TO THE WIFE AND KIDDIES, SO ONCE MORE FOR THE RECORD; THERE'S THIS 'CAPS LOCK' YOU ALLEGEDLY HAVE, BUT IT'S 'INVISIBLE'! - RIGHT SO FAR?
THE GREAT SWANNO

Although many keyboards have a teensy red or green light to indicate whether the CapsLock key is on or off, I prefer to look at the screen while I type, not at my fingers. Besides, I often use an older XT keyboard, which doesn't come with "running" lights. I therefore decided to write a program to display the CapsLock key state where my eyes are focused—on the screen.

Because the code would have to work with other software, a Terminate and Stay Resident (TSR) approach seemed best. The question was: Where should I display the information? How could I avoid conflicts with other programs that also write to the screen? As it so often turns out, the answers to these questions had been staring me right in the face.

But of Course—the Cursor

My solution to the "Case of the Invisible CapsLock" is to use the IBM BIOS Video I/O interrupt (number 10 hex) to change the cursor shape. A thin underline cursor, its normal look, indicates the CapsLock key is off. A half height, fat cursor indicates CapsLock is on and all typing is in uppercase. This plan works with most programs, doesn't interfere with text on display, and always correctly shows the CapsLock setting where my eyes usually are focused—at the cursor location.

Although it looks solid, the cursor is composed of lines, one on top of the other, numbered from the top starting with zero. To change the cursor shape, execute the instruction, INT 10h, with register AH equal to 1, selecting the BIOS interrupt's "Set-Cursor Type" option. Pass the cursor's start-line number in CH and end-line in CL. The start-line must be less than or equal to the end; otherwise, weird-looking cursor patterns can appear. As an example, the following assembly language fragment changes the cursor to a full height block.

```
mov  ah,1      ; select cursor style
mov  ch,0      ; start line
mov  cl,13     ; end line
int  10h       ; execute video i/o interrupt
```

Unfortunately, because different video modes use different character matrix dimensions, the code makes the cursor disappear in some graphics displays. This happens because the hardware requires the cursor's start and end lines to be in the same range as the current video mode's character matrix. Before changing the cursor, then, it's a good idea first to examine the video mode and then select appropriate start- and end-line values. Monochrome video uses a 9 x 14 character box; 350-line EGA graphics use 8 x 14; while CGA and PCjr graphics use 8 x 8. The CapsLock program takes these differences into account. (If you have trouble with newer video "standards," though, please let me know.)

Installing the CAPSLOCK program

To install the program in your system, type in the accompanying program listing and save as CAPSLOCK.ASM. Assemble with Microsoft's Macro Assembler (MASM) and link with these commands:

```
MASM CAPSLOCK;
LINK CAPSLOCK;
EXE2BIN CAPSLOCK
RENAME CAPSLOCK.BIN CAPSLOCK.COM
```

Ignore the linker message, "Warning: no stack segment." Run the program to install the TSR code in memory or, even better, add CAPSLOCK to your AUTOEXEC.BAT file to load the program every time you boot. It's probably best to install CAPSLOCK before other TSR programs like SideKick that completely take over the keyboard; otherwise, the cursor might not change shape properly when you are typing in a TSR window.

To keep the program short, I did not design it to check whether it is already installed, a typical TSR characteristic. In other words, if you run CAPSLOCK twice, two copies of the program end up in memory. If this bothers you, see the November/December 1986, and March/April 1987, issues of *PJ* for several articles that suggest ways to avoid accidental reinstallation of TSR routines. I prefer not to add such checks, opting instead for the smallest possible code file. Installing the program via AUTOEXEC.BAT also helps avoid accidentally running it twice.

Inside CAPSLOCK

The program begins with several equates. Labels FATSTART14 to ENDLINE8 specify the start- and end-line values for 14- and 8-line cursors. Change these values to alter the cursor pattern, making it taller or shorter as you prefer.

INT9 is the TSR code. After installation, it runs twice for every keypress—once when you press down a key (make) and once when you release it (break). The routine works by saving a copy of the current CapsLock key setting, a single bit in the BIOS data segment keyboard flag byte (kbflag). The first action is to test if this setting has changed since the last time. If not, the routine immediately exits without changing the cursor shape.

If it detects a change in the CapsLock bit, INT9 saves the new setting in a local byte variable, OLDCAPS. It then loads the current video mode (CRTMODE) into AL, jumping either to label LINE14 or LINE8 to set up 14- or 8-line cursor values. After that,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the three instructions at CHANGE: test the CapsLock bit, now in AH, selecting either a fat or thin pattern. Then interrupt 10 hex changes the cursor shape.

The end of INT9 may look odd if this is your first time through a TSR program. The DB and DD directives insert an uninitialized far jump at the end of the resident code. In the initialization part of the program at label INT1:, you'll find two instructions that insert the current interrupt 9 keypress handler vector into this far jump's address field. This way, the new code chains to the original pre-installation interrupt routine. Any number of TSR programs can hook into this and other interrupts in a similar fashion.

One portion of the initialization code, beginning with ASSUME ds:BIOSDATA, turns off the CapsLock bit, regardless of its current setting. If you install the program via AUTOEXEC.BAT, this is not necessary because booting also turns off the CapsLock bit. If you install it manually, though, and you have previously pressed CapsLock, the cursor will not assume the correct shape until you press CapsLock again.

The final two instructions (before the program variables) end the installation with the DOS "Terminate and Stay Resident" interrupt 27 hex. Assigning the address of INIT to DX tells DOS to permanently keep in memory all bytes below that point (at lower addresses). This retains the new interrupt code while throwing out the installer's initialization part and string variables.

Side Benefits and Difficulties

You might have trouble using CAPSLOCK with programs like BASIC that change cursor shapes to indicate various modes and options. Using CAPSLOCK probably won't affect the way such programs operate, but it might cause the cursor to assume the wrong shapes at the wrong times.

After using CAPSLOCK for several months with DOS versions from 2.0 through 3.2 plus a variety of software packages, I've noticed it has two side benefits. Pressing the CapsLock key two or three times changes the cursor shape back to normal in programs that permanently reset it to a full-height block, which looks obtrusive to me. (By modifying the start- and end-line constants, you can do the reverse, normally displaying a fat cursor if that's your preference and shrinking to half height with the CapsLock key down.)

The second side benefit is for those times when no matter how hard I stare at the screen, I cannot find the cursor. This is especially aggravating when reading through program listings with many underlines, all of which look like cursors—especially late at night when my eyes aren't focusing too well. Pressing CapsLock fattens the little flasher, making it easy to pick it out of a complicated display.

I've found CAPSLOCK to be a simple though useful addition to my system. It's especially helpful on PC and XT keyboards without CapsLock lights and lets me keep my eyes where they belong—on my work instead of my fingers. □

Tom Swan is the author of several books including Mastering Turbo Pascal *and* Mastering Turbo Pascal Files *(fourth quarter 1987) published by Howard W. Sams, Inc.*

Code follows.

Unbelievable!

SOURCER™

Creates commented source code and listings from memory, .COM files or .EXE files

- **CLARIFY UNDOCUMENTED CODE**
- **EASILY MODIFY PROGRAMS**
- **RECREATE LOST SOURCE CODE**
- **DOCUMENT FOREIGN CODE IN ENGLISH**

SOURCER™ creates detailed commented listings and source code directly suitable for assembly. Built in data analysis and simulator resolves multiple data segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Supports 8088 through 80286, and V20/30.

BIOS SOURCE

PS/2 ▪ AT ▪ XT ▪ PC ▪ Clones

- **CHANGE & ADD FEATURES TO YOUR BIOS**
- **CLARIFIES BIOS INTERFACES**
- **SPECIFIC TO YOUR MACHINE**

The bios pre-processor to SOURCER™ provides the first means to obtain accurate legal source listings for any bios! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive data labels such as "video_mode" and "keybd_q_head," and much more.

SOURCER™	$ 99.95*
SOURCER™ with bios pre-processor	$139.95*

(*OUTSIDE USA, ADD $15 SHIPPING; CA RESIDENTS, ADD SALES TAX)

To order or receive additional information just call!

800-538-8157 x 811 (outside Calif.)
800-672-3470 x 811 (inside Calif.)

V COMMUNICATIONS
3031 Tisch Way, Suite 200, Dept. J1
San Jose, CA 95128
(408) 296-4224

Number **125** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PORTABLE POWER WITH MasterFORTH

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest.

If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in assembler to make it even faster. MasterFORTH's relocatable utilities and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint and trace your way through most programming problems. A string package, file interface and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

CP/M™

The package exactly matches *Mastering Forth* (Brady, 1984) and meets all provisions of the Forth-83 Standard.

MasterFORTH standard package	$125
(Commodore 64 with graphics)	$100
Extensions	
Floating Point	$60
Graphics (selected systems)	$60
Module relocator (with utility sources)	$60
TAGS (Target Applic. Generation System) – MasterFORTH, target compiler and relocator	$495
Publications & Application Models	
Printed source listings (each)	$35
Forth-83 International Standard	$15
Model Library, Volumes 1-3 (each)	$40

(213) 821-4340

MICROMOTION

8726 S. Sepulveda Bl., #A171
Los Angeles, CA 90045

Number **13** on the Reader Service Card

Code from "CapsLock" by Tom Swan

1

```
;-----------------------------------------------------------------------
;
;  PURPOSE      : CapsLock Cursor Display program (TSR)
;
;  SYSTEM       : IBM PC-DOS
;
;  LANGUAGE     : Microsoft Macro Assembler (MASM)
;
;  AUTHOR       : (C) 1987 by Tom Swan, Swan Software, P.O. Box
;                 206, Lititz, PA 17543
;
;-----------------------------------------------------------------------
;----- Equates

cr              equ     13              ;ASCII carriage return
lf              equ     10              ;ASCII line feed
caps            equ     040h            ;CapsLock kbflag bit
monochrome      equ     7               ;Monochrome video mode
last8x8mode     equ     0eh             ;If <=, assume 8x8 chars
fatStart14      equ     7               ;14-line cursor fat
thinStart14     equ     11              ; and thin start/end lines
endLine14       equ     12
fatStart8       equ     3               ;8-line cursor fat
thinStart8      equ     6               ; and thin start/end lines
endLine8        equ     7


BIOSDATA        SEGMENT AT 40H

kbflag          equ     017h            ;Keyboard flag offset
crtmode         equ     049h            ;Current video mode
BIOSDATA        ENDS


;----- Start of Program

CSEG            SEGMENT byte
                ASSUME  cs:CSEG,ds:CSEG,ss:CSEG,es:CSEG
                ORG     100h            ;com file entry point

CapsLock:       jmp     short init      ;jump to initialize program


page;-----------------------------------------------------------
; INT9
;-----------------------------------------------------------
; Purpose       -- Trap keyboard interrupt #9 on every
;                  keypress.  Set cursor according to
;                  current CapsLock/NumLock settings
;                  then continue with normal interrupt.
;
; Called by     -- Keyboard hardware interrupt (#9)
;
; Calls         -- Jumps to pre-installation code
;                  Does not call DOS
;
; Arguments     -- none
;
; Registers     -- none changed
;-----------------------------------------------------------

        ASSUME  cs:CSEG, ds:BIOSDATA

oldcaps db      0               ;last known capslock setting
int9    PROC    near

        pushf                   ; save flags & registers
        push    ax
        push    cx
        push    ds

        mov     ax,BIOSDATA     ;set dsregister to
        mov     ds,ax           ; BIOS data segment
        mov     ah,byte ptr [ds:kbflag]         ;get keyboard flag
        and     ah,caps         ;isolate capslock bit
        cmp     ah,byte ptr [cs:oldcaps]        ;compare old setting
        je      exit            ;exit if not changed
        mov     byte ptr[cs:oldcaps],ah         ;save new setting
        mov     al,byte ptr [ds:crtmode]        ;get video mode
        cmp     al,monochrome   ;if monochrome
        je      line14          ; then use 14-line cursor settings
        cmp     al,last8x8mode  ;else if less than this
        jle     line8           ; then use 8-line cursor settings

line14:
        mov     al,fatStart14   ; 14-line cursors
        mov     ch,thinStart14
        mov     cl,endLine14
        jmp     short change

line8:
        mov     al,fatStart8    ; 8-line cursors
        mov     ch,thinStart8
        mov     cl,endLine8


;----- Change cursor style.  At this point, ah=capslock bit,
;      al=fat start line, ch=thin startline, cl=end line
change:
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        or      ah,ah           ;test capslock setting
        jz      nocaps          ;jump if key is off (thin style)
        mov     ch,al           ;else switch to fat style

nocaps:
        mov     ah,1            ;select cursor style
        int     10h
;execute BIOS video I/O interrupt

exit:
       pop     ds              ;restore registers & flags
       pop     cx
       pop     ax
       popf


;----- The following reserves space fora far jump instruction,
;      filled in by the initialization code.  This lets the program
;      continue to the original address that handled this interrupt
;      before installing the TSR code.

        db      0eah    ;ea=far jump hex code

address:
        dd      0       ;32-bit jump address (unitialized)

int9    ENDP            ;end of procedure


page
;--------------------------------------------------------
; INIT
;--------------------------------------------------------
; Purpose       -- Initialize interrupt trap, terminate
;                  program and stay resident.  All code
;                  from this point on is NOT saved
;                  in memory.  Everything before stays.
;
; Called by     -- Main program (via jmp)
;
; Calls         -- DOS
;--------------------------------------------------------

        ASSUME  cs:CSEG,ds:CSEG

init    PROC    near
        mov     dx,offset id    ;print program id
        mov     ah,9
        int     21h


;----- Get vector to current interrupt #9 routine

        mov     ax,3509h        ;ah=35
(get vector)
                                ;al=09
(vector number)
        int     21h             ;set es:bx to current vector


;----- Insert vector as far jump address at end of resident code

init1:  mov     word ptr address,bx     ;set far jmp offset address
        mov     word ptr address+2,es   ;set far jmp segment address
;----- Change interrupt vector to the new TSR routine.

        mov     dx,offset int9  ;dx=address of routine
        mov     ax,2509h        ;ah=25(set vector)

;al=09(vector number)
        int     21h             ;set interrupt 9 vector


;----- Make sure capslock bit is off.

        ASSUME  ds:BIOSDATA

        push    ds
        mov     ax,BIOSDATA
        mov     ds,ax           ;address BIOS data segment
        and     byte ptr [ds:kbflag],not caps   ; turn off caps
        pop     ds

        ASSUME  ds:CSEG


;----- Terminate and stay resident (TSR).

        mov     dx,offset init  ;dx = first available address
        int     27h             ;terminate, stay resident
init    ENDP                    ;end ofprocedure


;----- Program variables.

id      db      cr,lf, 'CapsLock Cursor Display Program'
        db      cr,lf, '(C) 1987 by TomSwan'
        db      cr,lf, 'Swan Software, PO Box 206, Lititz PA  17543'
        db      cr,lf, '$'

CSEG    ends                    ;end ofsegment
        end     CapsLock        ;end of program
```


#1 C interpreter

C-terp

The professional C development environment

Your C compiler creates great final code . . . but as a programming tool, it's too, t o o s l o w.

With C-terp you can edit, debug, and run without the wait. Nothing, but nothing, is faster for developing professional C programs.

Choose the perfect C-terp companion for your C compiler

C-terp/Microsoft	C-terp/XENIX
C-terp/Lattice	C-terp/Aztec
C-terp/Mark Williams	C-terp/C86

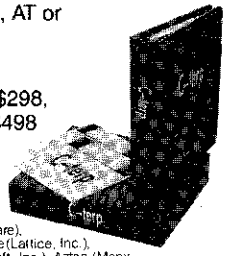
Link in all your compiler's functions, your own functions, add-on libraries, assembly routines, and data objects. Get instant access to everything in the C-terp interactive environment.

Only C-terp offers all this and more

- Full K&R with common ANSI enhancements
- Source level interactive debugging
- Software paging for your big jobs
- Complete multi-module support
- Run-time pointer checking
- Unsurpassed reconfigurable screen editor
- Dual display and full graphics support
- Large model
- Call-in

ORDER C-terp TODAY (specify compiler)

C-terp runs on IBM PC, AT or compatibles.



Price:
MS-DOS 2.x and up - $298,
Xenix System V 286 - $498
MC, VISA, COD
30-day money-back GUARANTEE

Trademarks: C-terp (Gimpel Software), C86 (Computer Innovations), Lattice(Lattice, Inc.), Xenix, Microsoft, MS-DOS (Microsoft, Inc.), Aztec (Manx Software), Mark Williams (Mark Williams Company), IBM (International Business Machines, Inc.)

GIMPEL SOFTWARE

3207 Hogarth Lane, Collegeville, PA 19426

(215) 584-4261

Contact this Advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TSRs Using Expanded Memory with Turbo Pascal

by
John J. Newlin

After I developed a Terminate and Stay Resident (TSR) program written in Turbo Pascal, it occurred to me that expanded memory might be the ideal environment from which to execute a TSR. Expanded memory for the IBM PC XT and clones is add-on memory accessed via a driver that conforms to the Lotus, Intel, Microsoft (LIM) Expanded Memory Specification (EMS). This article will explain a software design strategy that will enable a Turbo Pascal program to load into and execute from expanded memory.

The LIM specification was developed and promulgated jointly by Lotus Development, Intel, and Microsoft Corporations. This specification provides all the details necessary to write applications software designed to utilize expanded memory. It is available from any of the three companies and can also be found on many bulletin boards around the country, including CompuServe and Genie. I strongly advise anyone who contemplates interfacing software with an EMS driver to acquire a copy of the specification [see the accompanying article——Ed.].

An EMS driver is known as the Expanded Memory Manager (EMM). The EMM is usually installed as a device driver, and its job is to allocate and deallocate expanded memory "pages." When the EMM is installed, it is assigned a page frame that is normally one of the 64K byte memory segments above the 640K DOS limit as shown below:

Frame	Segment	Normal Use
A	A000h	EGA Video RAM
B	B000h	Video RAM
C	C000h	Hard disk BIOS
D	D000h	
E	E000h	AT Bios
F	F000h	PC & AT Bios

Some expanded memory drivers may use a frame segment that starts in one of the frames above and ends in the next higher frame; for example, one driver I know uses a frame segment of CC00H. It can be seen from the chart above that frame A is a candidate (if no EGA present), frame D is a candidate, and if the target machine is not an AT, frame E is a candidate. Each EMM frame is divided into four 16K pages of memory, and these pages are swapped by the EMM in and out of the frame as requested by applications programs written to utilize expanded memory.

Swapping is actually accomplished through page mapping registers that very rapidly assign one of the four 16K pages in the frame to a 16K memory bank on the EMS board. The page mapping registers are usually located in the same memory address space as BIOS ROM. Since ROM is read only memory and the page registers are write only, there is no conflict.

Since DOS currently does not support the loading and execution of programs above 640K in the memory space, a program that is to run in expanded memory must be either driven by a RAM resident kernel or loaded and executed by a custom designed loader. Clearly, the TSR kernel approach is much simpler to design and implement. The design strategy I chose for this project is outlined below in a set of high level requirements statements.

1. The main program shall reside in expanded memory and shall be written in Turbo Pascal.
2. The main program shall load itself into expanded memory and jettison any unneeded initialization code in the process.
3. The main program shall be activated from a TSR kernel that resides in normal RAM.
4. The TSR kernel shall be written in assembly language and shall occupy as little RAM as possible.
5. The TSR kernel shall conform to the LIM standard by saving and restoring the page mapping context upon entry and exit.
6. The TSR kernel shall contain a keyboard handler that operates through intercepts of calls to interrupt 16h, the BIOS keyboard interrupt vector.

EMM-Specific Procedures in the Main Program

The very first thing a program that intends to interface with expanded memory must do is to ascertain that an EMS driver is present. The LIM specification provides two methods for accomplishing this task. But before getting into the Turbo Pascal code that implements this task, it is necessary to define some data structures.

```
DEMO.INC sample include files

Type
  regpack     = record
                  case integer of
                    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
                    2 : (al,ah,bl,bh,cl,ch,dl,dh : byte);
                end;
  address     = record
                  offset,segment : integer;
                end;

Const
  activate       : integer = 0;
  demo_main      : integer = 0;
  emm_handle     : integer = 0;
  loaded_in_emm  : integer = $9876;
  emm_seg        : integer = 0;
  dataseg        : integer = 0;
  stackseg       : integer = 0;
  stackptr       : integer = 0;

Var
  vector : array[$00..$FF] of address;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For this project I placed all type, constant, and variable definitions in a single include file named DEMO.INC. The good reason for doing so will become more apparent later. The type **regpack** is used to employ the intrinsic Turbo procedures that give access to BIOS and DOS interrupts. **Address** is a type most useful in dealing with 8088 segment:offset addresses. Turbo typed constants are not really constants at all but are pre-initialized code segment variables. The typed constant **emm_handle** is placed in the code segment for reasons that will be explained later.

Now for the function that will report whether or not an EMS driver is present: Any EMM that meets the LIM specification will be installed to be accessed through int 67h. The LIM specification also requires that the EMM have the identifier string 'EMMXXXX0' located at offset 10 bytes from the int 67h handler segment value. To determine if an EMM driver is present, we need the segment address currently occupying the 67h slot of the interrupt vector table. That address is conveniently obtained through the use of the vector variable:

```
function emm_present : boolean;
var
  emm_id : array[1..8] of char;
begin
  emm_id := '        ';
  move(mem[vector[$67].segment:10],ems_str,8);
  emm_present := emm_id = 'EMMXXXX0';
end;
```

This function will return a value of true if an EMM is installed and a value of false if not.

Once it has been determined that an EMM is installed, the next step is to issue a request to the EMM for the desired number of expanded memory pages. Each application that uses expanded memory must do this, and if sufficient pages are available, the EMM will allocate those pages to the requesting program and issue that program a unique handle the program will use for all future communications with the EMM. The function **Get_handle** returns a true if the handle was issued and false if not. The reference parameter **handle** will return the actual EMM handle if the call was successful; otherwise, it will return the number of pages the EMM is reporting as available. Note that all of the EMM request functions call interrupt 67h and that all calls return a status code in AH. If the AH register returns a 0, the call was successful.

```
function get_handle(var handle : integer) : boolean;
var r : regpack;
begin
  r.ah := $43;
  r.bx := 8;
  intr($67,r);
  if r.ah = 0 then
    begin
      handle := r.dx;
      get_handle := true;
    end
  else
    begin
      r.ah := $42;
      intr($67,r);
      handle := r.bx;
      get_handle := false;
    end;
end;
```

If the EMM is not present in the system or reports insufficient expanded memory pages available, the main program will abort with an appropriate error message; otherwise, the main program can continue executing and use the acquired EMM handle (stored in **emm_handle**) to communicate with the EMM.

The next step is to query the EMM for the actual physical segment address of the applications frame in use by the EMM. The EMM maps logical pages (stored in memory on the EMM board) into physical pages located in the applications page frame assigned to the EMM. The following function returns the required segment address and, as explained above, this address will most probably be A000h, C000h, D000h, or E000h.

```
function emm_segment : integer;
begin
  r.ah := $41;
  intr($67,r);
  emm_segment := r.bx;
end;
```

Note that as in the case of **emm_handle**, **emm_segment** is a code segment variable.

One more EMM-specific procedure is needed in the main program, a procedure that will map the logical pages allocated by the EMM into the physical address space of the applications page frame segment. Applications programs are assigned logical pages by the EMM, and in order for the program to access its assigned memory, it must request that the EMM map its logical pages into physical pages. EMM-assigned logical pages are numbered sequentially starting from 0 up to one less than the number of pages allocated. There are four physical pages numbered 0 through 3. These pages occupy addressable memory as follows:

Page No.	Address Space
0	0000h - 3FFFh
1	4000h - 7FFFh
2	8000h - BFFFh
4	C000h - FFFFh

If the main program is large and complex and barely fits into the 64K frame segment, it might be necessary to jettison all the initialization code and data.

The address space values are offset from the page frame segment, and you will note that each page occupies 16K of addressable space. The applications program can use the assigned space for code or data, but must assume the responsibility for managing which page contains which data or code. Suppose, for example, that the page frame segment address in use is A000h and that the applications program desires to store an array of data in one of its assigned logical pages. If it mapped logical page 9 into physical page 4, it could store an array of up to 16K bytes at address A000h:C000h. Any time it needed data from this array, it would have to insure that its logical page 9 was mapped into physical page 3. Or the applications program could store an executable procedure in logical page 9 and call it after mapping that page into physical page 3. The following procedure will be used by the main program to map one of its logical pages into a physical page in the frame segment:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
procedure map_memory(logical,physical : integer);
var r : regpack;
begin
  r.ax := physical;
  r.ah := $44;
  r.bx := logical;
  r.dx := emm_handle;
  intr($67,r);
end;
```

In this procedure, the logical page number (owned by the main program) is placed in the BX register, the physical page number is placed in the AL register, and the assigned handle is placed in the DX register.

Move the Main Program, Data, and Stack into Expanded Memory

Now that the functions and procedures required for making expanded memory available are defined, the next design task is to deal with the problem of moving main program code, data, and stack into expanded memory. One of two approaches may be used for this task, and the approach selected depends upon several factors. One factor is the size of main program code, data, and stack. If the main program is relatively small, the entire program can simply be moved into expanded memory. If the main program is large and complex and barely fits into the 64K frame segment, it might be necessary to jettison all the initialization code and data.

For example, once the functions **emm_present**, **get_handle**, and **emm_segment** are executed, they are no longer needed by DEMO. You might have some data structures that are used by program initialization code but are no longer required after the program has executed. Functions/procedures that create sign-on screens, initialize program variables, and perform other housekeeping chores not needed by the main program more than once are candidates for leaving behind as excess baggage. Even though expanded memory is usually plentiful, it is prudent to impose as few memory page management functions upon the main program as possible, especially in the case of executable code.

The first step in jettisoning unwanted code and data is to declare a procedure named **dummy** as follows:

```
procedure dummy;
begin
end;

var
  lastx,lasty,lastvar : integer;
```

Place these declarations just following the last Turbo Pascal function/procedure in the main program that you want moved into expanded memory. Any other initialization variables may be declared anywhere in the program following the variable declarations above.

The next step is to calculate and store the new value of the DS register. The CS register will be set automatically when the main program is called by the TSR kernel, and the SS register will be set to the same value as the CS register. The following Turbo code calculates and stores DS and SS:

```
emm_seg := emm_segment;
lasty := ofs(dummy);
lastx := lasty shr 4;
dataseg := emm_seg + lastx + 16;
stackseg := emm_seg;
stackptr := $BFFF;
```

This code sets the variable **lasty** equal to the offset address of procedure **dummy**, which is beyond the end of the last function/procedure we desire to place in the main program code segment. Since we want to start the data segment at a point immediately following the code segment, this value is converted from an offset address to a segment address by shifting it left 4 times and adding it to the code segment value. Code segment overrun is not a problem because we know that the frame segment value is always set on a zero page boundary. 16 bytes are added to insure that no code/data segment overlap occurs. Then the stack segment is set equal to the code segment. It is not necessary to fool with the extra segment value since it is not critical in Turbo Pascal.

But the stack pointer is a different matter. The stack pointer value normally will be a function of how the Turbo options menu was set up prior to compiling. In this case, a simple solution is to set the stack pointer value ourselves and thus not have to be concerned about the options menu before compiling. In this example, a stack pointer value of BFFFh was chosen because there is ample space for the stack between the end of the data segment and the top of page 2 of the expanded memory frame segment. It is the programmer's responsibility to select a workable stack pointer value based upon the size and structure of the main program. The amount of stack needed for the main program will be a function of the largest amount of local variable space required by any of the main program functions/procedures. It is relatively simple to derive an empirical estimate of stack size needed by examining the main program listing. In Turbo Pascal, local variables not declared in the code segment are created as temporary variables on the stack when the function/procedure is called. If stack space becomes a problem due to a large local variable requirement, required stack space can be reduced by making some of the variables global. Just remember that the trade off is between total program code/data size and stack size.

The final task that the main program will accomplish before terminating will be to move the code and data segments up into its allocated expanded memory space. But first the allocated memory pages must be mapped into the EMM applications frame. The following code accomplishes that:

```
map_memory(0,0);
map_memory(1,1);
map_memory(2,2);
map_memory(3,3);
```

Movement of code and data is effected by the following statements:

```
move(mem[cseg:0],mem[emm_seg:0],ofs(dummy));
move(mem[dseg:0],mem[dataseg:0],ofs(lastvar));
```

The first statement physically moves the code in the main program into expanded memory. The second statement moves the data in the data segment. Notice that any code declared after the procedure **dummy** and any data declared after **lastvar** will not be moved and therefore jettisoned. Now main program code and data are in place in expanded memory, the segment registers and stack pointer values are set, and the program is ready to execute when called by the TSR kernel program. Later in this article I will discuss initializing pointers and other variables for main program use.

The TSR Kernel

In developing a design strategy for the TSR kernel, the first order of business is to select a keyboard handler. There are basically two ways to go here since there are two interrupts in the IBM PC that process keystrokes: INT 9h and INT 16h. Using INT 9h, the keyboard hardware interrupt, entails a more complex (and larger) handler as well as the trapping of other interrupts. An INT 16h, the ROM BIOS handler, is much simpler to design and safer to use overall.

Listing 1 is the assembler source code for the TSR kernel. Notice that there are 4 equates at the top of the listing. These

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

equates are derived from the following Turbo program:

```
EQUATE.PAS sample program

Program EQUATES;

type str4 = string[4];

function hex(n : integer) : str4;
var h : array[0..$F] of char;
begin
  h := '0123456789ABCDEF';
  hex := h[hi(n) div 16] + h[hi(n) mod 16] +
         h[lo(n) div 16] + h[lo(n) mod 16];
end;

{$I DEMO.VAR}

begin
  writeln('activate       equ ',hex(ofs(activate)),'H');
  writeln('demo_main      equ ',hex(ofs(demo_main)),'H');
  writeln('emm_handle     equ ',hex(ofs(emm_handle)),'H');
  writeln('loaded_in_emm  equ ',hex(ofs(loaded_in_emm)),'H');
end.
```

This program (EQUATE.PAS) will provide the correct values (in hex) of the variable offsets needed by the TSR kernel. It need be run only one time if the constant declarations in DEMO.VAR are not changed. The equate values that appear in this listing are derived using Turbo Pascal Version 3.01A. Other versions may produce different addresses, so be sure to compile and run EQUATES.PAS to assure that the values are correct for your version of the TSR kernel.

A jump directly to the initialization code starts the kernel program. After the DS register is set, a call to interrupt 16h with AH set to 9Bh is made. This demonstrates a clean, safe method of checking to see if the kernel has already been loaded. If the kernel is already installed, the call will return a value of ABCDh in the AX register and the program will abort without installing a second copy in RAM.

Next, a check is made to be sure that an EMM driver is present. The method used to do this is exactly the same as that used in the main program. If no EMM driver is installed, the program aborts with an appropriate error message. If an EMM driver is present, a check is made to determine if the main program is loaded into the EMM applications frame. If not, the program aborts.

The applications frame segment address is acquired from the EMM, and using this and the pre-determined main program addresses coded in the equates statements, the kernel acquires the hot key value (in this case Alt-F10), the EMM handle, and the offset of the main program's handler procedure.

Continued...

DON'T LIKE YOUR DINNER?
WHY, JOHNNY, WHEN I WAS YOUR AGE WE HAD TO EAT EVERYTHING ON OUR PLATES EVEN WHEN IT WAS NOTHING. WHICH IT OFTEN WAS, AND THIS WAS AFTER A SEVEN MILE WALK HOME FROM SCHOOL WITH SNOW OVER OUR HEADS AND THIS WAS IN SOUTHERN CALIFORNIA AND WHEN I WAS YOUR AGE, EVEN

ONE OF JOHN NEWLIN'S EARLY BRUSHES WITH EXPANDED MEMORY.

Create 68K Embedded Systems On ATs

Oregon Software Brings VAX and VME Pascal-2® Cross-Development Tools To MS-DOS Workstations

Get All Your Tools In One Package

The Pascal-2 Cross-Development System gives you price/performance value that beats high-priced workstations. Compare these features and call for further information.

Compiler

- 68000, 68020, and 68881 instruction sets
- ROMable code
- reentrant code
- separate program sections for code and data
- easy access to hardware through fixed memory locations (I/O space)

Assembler-Linker

- VERSAdos compatible output
- easily linked, relocatable S-record format

Concurrent Program Support

- process synchronization primitives for creating a multiprocessing real-time kernel
- interactive post-mortem analyzer
- software-selectable interrupt levels

Stand-Alone Library

- source supplied: users may adapt code as needed
- user programs that run without an operating system or other real-time kernel
- compact code for extremely small ROM programs or limited memory devices
- ability to write device drivers in Pascal or assembly code
- user-customized system services or control programs
- easily-gathered performance analysis statistics

Source Management

- a version-control system to track and identify each phase of development
- optional branching to multiple versions from a single source
- automated rebuilding of programs from component modules (a Unix-style Make function)
- a restricted-access system for coordinating revisions

OREGON SOFTWARE

800-367-2202

6915 SW Macadam Avenue, Portland, Oregon 97219 U.S.A.
503-245-2202 FAX 503-245-8449 TWX 910-464-4779

The following are trademarks: Pascal-2, SourceTools, and Oregon Software, Oregon Software, Inc., VAX and VMS Digital Equipment Corporation, UNIX, AT&T Bell Laboratories, Incorporated. MS, and MS-DOS are registered trademarks of Microsoft Corporation.

Number **102** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Finally, the kernel uses DOS interrupt 21h function calls 35h and 25h to install the interrupt 16h handler. Function call 35h is made to acquire the current interrupt 16h segment and offset addresses, and these are saved in local code segment variables. Then function call 25h is made to install the kernel handler. A sign-on message is sent to the screen, and then interrupt 27h is invoked to make the kernel memory resident.

Once the main program is installed in expanded memory and the kernel TSR is in place, pressing the hot key combination (Alt-F10) will cause the kernel interrupt 16h handler to pass control to the main program via procedure **run_demo**. I won't dwell on the INT16 procedure except to point out that it intercepts calls requesting a keystroke from the keyboard buffer and calls requesting the status of the buffer. If the hot key is detected, **run_demo** is called.

The first thing that **run_demo** does is to call procedure **save_map**. The LIM EMS provides two function calls for the use of interrupt driven programs. **save_map** calls the EMM with a request to save the current expanded memory page mapping context. This is important since our hot key might be pressed when an applications program has pages of data mapped into the applications frame. It will not know that it has been interrupted, and if the page mapping context is not saved and restored, our main program code will be present in the applications frame instead of the interrupted program's data.

Following the call to save the mapping context is a call to place our main program's code into the applications frame. A local flag indicating that our program is active is then set. This prevents a re-entrant call to our program while it is executing. Next, the AX register is saved locally because it is used in the process of switching from the caller's stack to a local stack.

Switching stacks is accomplished by first saving the caller's SS and SP registers in local variables and then loading SS and SP with local values. Note that stack space has been reserved through the declaration of the local variable **local_stack** and that the stack pointer is loaded with the offset of the variable **stkptr**. Then the caller's registers are saved on the local stack and a far call is made to our code in expanded memory. The far call is forced by declaring the label **demo** as a double word. During kernel initialization, **demo_ofs** was loaded with the offset of the main program's handler and **demo_seg** was loaded with the EMM frame segment.

When the main program has finished its task, control is returned to the point just following the far call. Here the caller's registers are restored from the local stack, the stacks are switched again, and the active flag is reset. The last thing accomplished in this procedure is to call the EMM to restore the page mapping context. Whatever pages were active in the applications frame when the interrupt occurred should once again be present, and any program using these pages should be able to go on about its business as if nothing had occurred.

A return to the keyboard handler is executed where the keyboard buffer is flushed and control returned to the interrupted applications program or DOS, whichever the case might be.

Procedure handler

At this point, half the job is finished. We have designed and coded a kernel TSR that will load into memory, monitor the

$99 QUICK DATA BASE

QuickBase uses simple menu selections to guide you thru quick database setup, record entry & update, report design & output (10 levels of sort/select). Track clients & prospects, setup mail labels & mailmerge form letters, bill of materials, etc.

$29 CASINO CARD GAMES

Just like you've seen in Vegas and Atlantic City! You get BlackJack, Slot Poker, and a "bluffing" game of Draw Poker! BlackJack allows play variations & tracks statistics. Super card graphics & windows. CGA or Mono.

$59 POPUP TIME & BILLING

InsideTrac records billable and non-billable time and expenses for attorneys, CPAs, and consultants. Different billing rates for multiple employees, detail & summary time period billing reports, built in help manual and popup menu operations.

HAWKEYE GRAFIX Inc
BOX 1400, OLDSMAR FLORIDA 33557
DIAL 813-855-5846

Number **57** on the Reader Service Card

Manipulate files by date

Ever wanted to copy **only** those files you changed in the last two days? Or do you need to delete **last** month's data files? Maybe you have to copy all files created last year to a backup disk. If so you need **THE DATE UTILITIES!** With these routines you can copy, erase, or display files created **on, before, or after** a specified date. All utilities support paths, full **wildcards** for file names, and indirect files for dates.

- ☐ **DCOPY** copies files based their name **and** date. It won't copy an old file on top of a new one and offers an optional prompt before copying. DCOPY pauses when a floppy becomes full and lets you insert a new floppy or format a floppy before continuing.
- ☐ **DERASE** lets you delete a file or group of files based on their name and date. A prompt before erase mode is also offered.
- ☐ **DDIR** displays a disk directory limited by file name and date. The double column directory can be sorted based on name, extension, date, or file size.
- ☐ **CDATE** changes the date and time stamp of a file. An optional prompt before change mode is also offered.

Version 1.4 of THE DATE UTILITIES package costs $47.50. No charge for shipping in the U.S. These routines are **not** copy protected and come with bound typeset documentation. IBM PC or any compatible, MS-DOS 2.1 & up. 30 day money back guaranty!

VISA 312-377-3520 MC

ECI P.O. Box 61, Wasco, Illinois 60183

Number **70** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

keyboard for a specific keystroke combination, and call a main program that is resident in expanded memory. Let's go back to the main program now and examine the procedure that will be called from the kernel TSR. Procedure handler in the program listing for DEMO.PAS is a combination of Turbo inline and normal code. It contains a prolog and epilog written in inline code that sandwiches the normal code. Normally this procedure will be the last procedure before procedure dummy in the main program.

The first two bytes of the prolog undo the two push BP instructions compiled into all Turbo procedures. After the interrupts are disabled, the caller's segment registers are saved locally and replaced by the main program segment register values that were saved when the main program loaded and moved itself into expanded memory. We have access to these values because they are stored in the Turbo code segment. This is important because when the handler gets control, the only register value that is known is the CS register. That is why these values are declared as Turbo typed constants. Then the stack is switched and the handler procedure is ready to call other procedures in the main program.

When the Turbo inline code comprising the epilog is executed, the saved segment registers are restored, the stack is switched back, and a far return is executed to return control to our kernel TSR. This interface is fairly compact and provides the means by which executable code residing in expanded memory may be accessed via a RAM resident kernel.

The program DEMO.PAS provides an example of how expanded memory might be exploited by the main program. DEMO.PAS doesn't really do very much, but I think it serves as an excellent illustration of how to control and use the assigned pages of memory. When the hot key (Alt-F10) is pressed, the main program handler procedure receives control via the kernel. The handler procedure then calls **save_screen** and **save_cursor**. In this example there is no functional reason to store the video buffer in logical page 5; rather, it is stored there to illustrate how a program may use the page mapping procedure to store data in different assigned logical pages.

Procedure **main_proc** is called next, and it makes several calls to the EMM to illustrate what information is available. When **main_proc** has completed its work, control is returned to the handler, which restores the original screen, restores the cursor size and location, and exits back to the TSR kernel. Procedures **print** and **printline** are not called but are included as examples of how one may effect printer I/O from Turbo Pascal without running into the DOS re-entrancy problem.

The DOS Re-entrancy Problem

The only Turbo I/O call from an interrupt driven, background resident program that can get you into trouble is the printer I/O call. It has been generally, and erroneously, stated many times that DOS is not re-entrant. This is not globally true of all DOS functions. Re-entrancy problems with DOS arise because DOS may use any one of three different stacks. DOS interrupts 25h and 26h (read and write disk sectors) use one stack, DOS interrupt 21h, function calls 1 to 0Ch use another, and function calls 0Dh on up use yet a third. When a memory resident program is loaded, DOS is using the third stack. If that program then makes a call to DOS that causes DOS to switch

WINDOWS – MENUS – DATA ENTRY – SCREENS

HI-SCREEN XL™

Application Developers! HI-SCREEN XL™, the newest version of HIGH SCREEN, is a complete and unmatched programming tool for managing your user interface. HI-SCREEN XL™ features a revamped and enhanced screen editor, increased speed and performance, along with a new thorough reference and user's manual.

Whatever tool you own: compare, you'll be convinced!

3. **DATA ENTRY**
Just define your data entry fields and let HIGH SCREEN™ do the job: -integer/real/alphabetical/date/chosen characters. format/range/upper case-lower case convertion/justify left/right/center/size/color. required input/auto-tabbing/password/Help message. Automatic error handling. Cursor and arrow keys management. Field by field or full screen checking mode for increased flexibility.

1. **WINDOWS:** for on-line help, menus, data entry. Up to 26 levels deep.
2. **MENUS:** pop-up, pull-down or Lotus-style, also for batch files.
3. **DATA ENTRY:** automatic field checking: format, type, range... field by field or full screen. Error handling.
4. **SCREENS:** full-featured screen editor. Language independent. Fast display from RAM.

Royalty free, not copy protected, 30 day money back guarantee, trade up available.

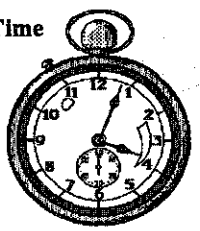
Softway, Inc.
PCSOFT Product Line
500 Sutter St., Suite 222
San Francisco, CA 94102

(415) 397-4668
HI-SCREEN XL™ is $149
(CA res. add tax), S&H USA $5
Visa, M/C welcome

PASCAL, C, dBASE, BASIC, COBOL, FORTRAN, ASSEMBLER, etc.

Number **110** on the Reader Service Card

Along With Your Computer, Your Time is the Most Important Thing You Own. . .So Why Waste It?



Quilt Programmer Productivity Tools will help you manage your software projects and get control of your time!

SRMS™
Software Revision Management System

- Full Featured Revision Control System
- All Versions stored in a Single ASCII File
- Support for Unlimited Libraries
- Support for all programming languages
- Allows you to use your current compilers and editors without conflict
- Windowing Shell interface to simplify use
- MERGE facility to consolidate different development paths easily, while pointing out conflicting areas
- Full audit trail tracking and reporting on all library components
- Handles big programming projects easily
- Full DOS Pathname and Environment Variable Support
- Requires DOS 2.1+, 224 K, F/H Disk

SRMS Version 3.0.....$185

QMAKE™
Intelligent Program Generation Utility

- Controls the rebuilding of even the most complex systems
- Relieves the developer of remembering which modules need to be rebuilt based on recent changes, how to rebuild them, and in what order to rebuild them
- Works with most compilers, assemblers, and linkers
- Supports full macro definitions, UNIX make-file compatability, recursive invocations, and command line parameters
- Interfaces completely with SRMS, providing you with a complete set of productivity tools to handle any size project
- Requires DOS 2.1+, 128K F/H Disk

QMAKE Version 1.2.....$99

SRMS + QMAKE $250

NEW ! TXT Tools NEW !

QSE - Quilt Text Stream Editor
QSRCH - Quilt File Search Utility (Like UNIX GREP)
QDIFF - Quilt Windowing File Difference Utility

TXTTOOLS Version 1.0..... $85

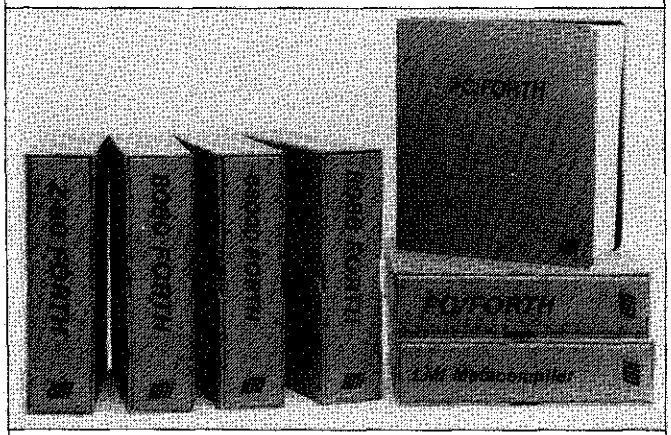
QUILT COMPUTING
7048 Stratford Road
Woodbury, MN 55125
(612) 739-4650

Volume Discounts and Dealer Inquiries Welcome

Number **46** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOTAL CONTROL
with *LMI FORTH*™



For Programming Professionals:
an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development: Interactive Forth-83 Interpreter/Compilers
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler
- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:
- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Number **63** on the Reader Service Card

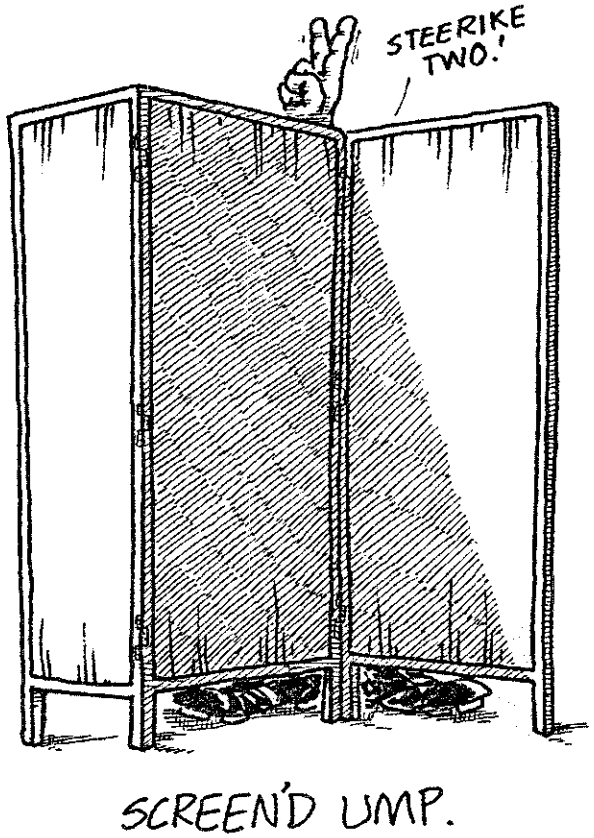
to the first or second stack, DOS will be lost when it regains control and the system will hang up.

Strangely enough, the only function call invoked by Turbo Pascal that causes a DOS stack switch is interrupt 21h, function 05h, DOS printer output. All other Turbo I/O calls are to the high numbered functions. One solution to this problem is to use procedures similar to **print** and **printline** that send output to the printer directly via BIOS interrupt 17h. Another solution is for the TSR to save and restore the caller's stack. I find the first method cleaner and therefore preferable to the second.

Compatibility

You might be wondering just how compatible DEMO.COM and KERNEL. COM are with other programs that use expanded memory. A good test of compatibility is to add some DOS disk I/O routines to DEMO and operate it in the same EMM environment as one of the popular disk caching programs. If everything is as it should be, when DEMO issues a call to read or write to the disk through DOS, the cache program will intercept the call and page its assigned EMS memory into the applications frame. If it's correctly designed, the cache program will issue calls to save and restore the EMM page mapping context before and after using expanded memory for the cache so that when DOS returns control to DEMO its code is back in the proper EMM page.

To give DEMO a test on your own system, first create DEMO.VAR and then EQUATES.PAS. Compile and run EQUATES.PAS, and record the output offset values for coding as equates in KERNEL.ASM. Then create KERNEL.ASM, assemble and link it, and convert it to a .COM file with EXE2BIN. Finally, compile DEMO.PAS to an executable disk file. Run



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Lotus/Intel/Microsoft EMS Functions

EMS general calling sequence is based on loading the AH register with a function code and executing an INT 67h. Other registers may also be used depending on the function invoked. Status is returned in AH with 0 returned for no errors. The Lotus/Intel/Microsoft (LIMS) Expanded Memory Specification Version 3.20 (dated September 1985) supports the following functions:

```
Function 1: Get Status
     Entry:    AH = 40h
     Returns:  AH = status

Function 2: Get Page Frame Address
     Entry:    AH = 41h
     Returns:  AH = status
               BX = segment address of page frame

Function 3: Get Unallocated Page Count
     Entry:    AH = 42h
     Returns:  AH = status
               BX = number of pages available
               DX = total number of pages

Function 4: Allocate Pages
     Entry:    AH = 43h
               BX = number of pages to allocate
     Returns:  AH = status
               DX = EMM handle

Function 5: Map Handle Page
     Entry:    AH = 44h
               DX = EMM handle (from function 4)
               BX = logical page to be mapped
               AL = physical page (0-3) destination
     Returns:  AH = status

Function 6: Deallocate Pages
     Entry:    AH = 45h
               DX = EMM handle (from function 4)
     Returns:  AH = status

Function 7: Get EMM Version
     Entry:    AH = 46h
     Returns:  AH = status
               AL = version number in BCD

Function 8: Save Page Map
     Entry:    AH = 47h
               DX = EMM handle
     Returns:  AH = status

Function 9: Restore Page Map
     Entry:    AH = 48h
               DX = EMM handle
     Returns:  AH = status

Function 10: Reserved
          (previously retrieved page mapping register I/O array)
     Entry:    AH = 49h

Function 11: Reserved
          (previously retrieved logical to physical page
          translation array)
     Entry:    AH = 4Ah

Function 12: Get EMM Handle Count
     Entry:    AH = 4Bh
     Returns:  AH = status
               BX = number of active EMM handles

Function 13: Get EMM Handle Pages
     Entry:    AH = 4Ch
               DX = EMM handle
     Returns:  AH = status
               DX = number of pages allocated to EMM handle

Function 14: Get All EMM Handle Pages
     Entry:    AH = 4Dh
               ES:DI = pointer to memory for array where a copy of
                       all active EMM handles and number of pages
                       allocated to each will be stored.  Array
                       entries are active EMM handle (word) followed
                       by number of allocated pages (word) totaling 4
                       bytes/EMM handle.
     Returns:  AH = status
               BX = number of active EMM Handles

Function 15: Get Page Map
     Entry:    AH = 4Eh
               AL = 0
               ES:DI = pointer to memory for destination array
     Returns:  AH = status
               ES:DI = pointer to memory area where the state of all
                       the mapping registers has been stored

Function 15: Set Page Map
     Entry:    AH = 4Eh
               AL = 1
               DS:SI = pointer to memory for source array where state
                       of all mapping registers has been stored
     Returns:  AH = status
               ES:DI = pointer to memory area where the state of all
                       the mapping registers has been stored

Function 15: Get and Set Page Map
     Entry:    AH = 4Eh
               AL = 2
               ES:DI = pointer to memory for destination array
               DS:SI = pointer to memory for source array where state of
                       all mapping registers has been stored
     Returns:  AH = status
               ES:DI = pointer to memory area where the state of all
                       the mapping registers has been stored

Function 15: Get Context Array Size
     Entry:    AH = 4Eh
               AL = 3
     Returns:  AH = status
               AL = size in bytes of page map array required for Get,
                    Set, or Get and Set Page Map functions mapping
                    registers has been stored
```

Error status values common to all functions have the following meaning:

```
AH = 80h  The EMM manager detected a malfunction in the EMM software
AH = 81h  The EMM manager detected a malfunction in the EMM hardware
AH = 84h  The function code passed in AH is not defined
```

Other status values may be returned indicating specific errors based on the function invoked

```
          AH = 83h  EMM handle specified could not be found
AH = 86h  EMM detected a "save" or "restore" page mapping context error
AH = 87h  Not enough expanded memory pages to satisfy your program's
          request
AH = 88h  Not enough unallocated pages to satisfy your program's
          request
AH = 89h  Can't allocate zero (0) pages
```

The Lotus/Intel/Microsoft Expanded Memory Specification changes from time to time, so you should contact Intel occasionally to insure that you are using the most recent version. Intel Customer Support can be reached at (800) 538-3373 weekdays from 7 a.m. to 5 p.m. Pacific Time. If you are in Oregon or outside the continental U.S., Intel can be reached at (503) 629-7354. The Canadian support number is (800) 234-0444.

The AST/Quadram/Ashton-Tate Enhanced Expanded Memory Specification (AQA EEMS) is a superset of the LIMS specification. Copies of the EEMS specification may be obtained from AST at (800) 527-7177. □

DEMO.COM first, and then immediately run KERNEL.COM. Activate DEMO by pressing the Alt and F-10 keys simultaneously. Note that the hot key combination can be changed by changing the value for the typed constant activate. For example, to change the hot key to Alt-F9, assign the value of 112 to activate.

I have verified that DEMO can be successfully operated with a number of commercially available programs that use expanded memory. I have also verified that any of the DOS int 21h functions from 0Dh on up can be issued from DEMO without any problem. Thus the strategy employed in designing and implementing a Turbo Pascal program that resides in expanded memory and is controlled through a TSR kernel is a powerful and efficient method of utilizing expanded memory. □

John J. Newlin is a retired Naval Aviator who currently holds a position with Hughes Aircraft Co. as a systems engineer. He holds degrees in mathematics and computer science and has programmed extensively in assembly, Turbo Pascal, Marshal Pascal, and Modula 2. He is the author of SCOUT, a memory resident disk/file/directory manager (reviewed by PC Magazine, *February 24, 1987), and is a member of the Association of Shareware Professionals. Readers who wish to contact Mr. Newlin may write to him at the following address:*

John J. Newlin
4060-228 Rosenda Court
San Diego, CA 92122

Code follows.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Minimize Turbo Pascal compile time.

Dramatically reduce debugging time.

Tmark allows Turbo Pascal to continue compiling without returning to line 1.

Tmark is a utility that fixes an annoying limitation of Turbo Pascal. Normally, every time Turbo finds an error while compiling, it must start compiling again from the very beginning. Tmark changes that. It allows Turbo to resume compiling from where you fixed the error, not back at line 1.

Tmark runs Turbo Pascal as a child process. This allows Tmark to monitor Turbo as it compiles your program. When Tmark detects that Turbo is about to issue an error message, it saves, on disk, an image of the memory and registers used by Turbo Pascal up to that point. Once you have fixed the error, Tmark will restore Turbo to its state before the error was found. Turbo will then resume compiling—from the fix.

Tmark will also save up to 7 images at designated lines in a program. When Turbo encounters a line which begins with {TMARK}, Tmark will save an image. You can resume compilation at any of these lines.

Don't lose another minute of compiling time, order Tmark now!

- Not memory resident
- Does not take over any software interrupts. If Turbo Pascal runs on your system, so will Tmark.
- Not copy protected

Tmark™

Only $80

Visa/MC accepted. Add $2 for shipping in U.S., $5 overseas

Tangent Designs, Inc.

P. O. Box 896
Lake Forest, IL 60045
(800) 356-2750 *Outside Illinois*
(312) 295-0030

Turbo Pascal is a registered trademark of Borland International. Supports Turbo versions 3.01 and above, with 8087 or BCD.

Number **126** on the Reader Service Card


```
DEMO.VAR include file
Copyright 1987 by John J. Newlin, 4060-228 Rosenda Court, San
Diego, CA 92122

(*  Type and variable declarations to
    be used with DEMO.PAS.              *)

type
  cursor_type = (normal,none);
  str4 = string[4];
  str255 = string[255];
  regpack     = record case integer of
                  1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
                  2 : (al,ah,bl,bh,cl,ch,dl,dh : byte);
                end;
  address = record
              offset,segment : integer;
            end;
  buffer_type = array[1..4000] of byte;
  hex_char_type = array[0..15] of char;

const
  activate      : byte = 113;
  dataseg       : integer = 0;
  stackseg      : integer = 0;
  stackptr      : integer = 0;
  demo_main     : integer = 0;
  loaded_in_emm : integer = $9876;
  emm_handle    : integer = 0;
  saveds        : integer = 0;
  savees        : integer = 0;
  savess        : integer = 0;
  savesp        : integer = 0;
  hex_char      : hex_char_type = '0123456789ABCDEF';
var
  vector : array[0..$FF] of address absolute $0:$0;
  emm_seg,cursor_size,cursor_posit,buffer_loc : integer;
  color,scan : boolean;
  main_buffer,aux_buffer : ^buffer_type;
```

```
DEMO.PAS
Copyright 1987 by John J. Newlin, 4060-228 Rosenda Court, San
Diego, CA 92122

{$I demo.var}
{$I-,K-,C-,V-,B-,F4}
(*  Expanded memory demonstration program
    Copyright 1987 by John J. Newlin
    All rights reserved                    *)

function emm_version : str4;
var x : integer;
    u,l : str4;
    r : regpack;
begin
  r.ah := $46;
  intr($67,r);
  x := r.al and $F0;
  x := x shr 4;
  str(x,u);
  x := r.al and $0F;
  str(x,l);
  emm_version := u + '.' + l;
end;

procedure emm_page_count(var total,allocated,free : integer);
var r : regpack;
begin
  r.ah := $42;
  intr($67,r);
  total := r.dx;
  free := r.bx;
  allocated := total - free;
end;

procedure save_cursor;
var r : regpack;
begin
  r.ah := 3;
  r.bh := 0;
  intr($10,r);
  cursor_size := r.cx;
  cursor_posit := r.dx;
end;

procedure restore_cursor;
var r : regpack;
begin
  r.cx := cursor_size;
  intr($10,r);
  r.ah := 2;
  r.dx := cursor_posit;
  intr($10,r);
  r.ah := $1A;
end;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

2

```
procedure map_memory(logical,physical : integer);
var r : regpack;
begin
  r.ax := physical;
  r.ah := $44;
  r.bx := logical;
  r.dx := emm_handle;
  intr($67,r);
end;

procedure printline(var str : str255);
var i : integer;
    r : regpack;
begin
  str := str + #13 + #10;
  for i := 1 to ord(str[0]) do
    begin
      r.ah := 0;
      r.dx := 0;
      r.al := ord(str[i]);
      intr($17,r);
    end;
end;

procedure print(var str : str255);
var i : integer;
    r : regpack;
begin
  for i := 1 to ord(str[0]) do
  begin
    r.ah := 0;
    r.dx := 0;
    r.al := ord(str[i]);
    intr($17,r);
  end;
end;

function hex(n : integer) : str4;
begin
  hex := hex_char[hi(n) div 16] + hex_char[hi(n) mod 16] +
         hex_char[lo(n) div 16] + hex_char[lo(n) mod 16];
end;

procedure save_screen;
begin
  map_memory(3,5);
  move(mem[buffer_loc:0],main_buffer^,4000);
  map_memory(3,3);
end;

procedure restore_screen;
begin
  map_memory(3,5);
  move(main_buffer^,mem[buffer_loc:0],4000);
  map_memory(3,3);
end;

function number_of_emm_handles : integer;
var r : regpack;
begin
  r.ah := $4B;
  intr($67,r);
  number_of_emm_handles := r.bx;
end;

function pages_assigned_to_handle(handle : integer) : integer;
var r : regpack;
begin
  r.ah := $4C;
  r.dx := handle;
  intr($67,r);
  pages_assigned_to_handle := r.bx;
end;
```

3

```
procedure main_proc;
var r : regpack;
    k : char;
    total,used,free,handles,pages,x,y : integer;
begin
  x := 19;
  y := 8;
  emm_page_count(total,used,free);
  handles := number_of_emm_handles;
  pages := pages_assigned_to_handle(emm_handle);
  clrscr;
  gotoxy(x,y);
  write('The Expanded Memory Manager version is ',emm_version);
  gotoxy(x,wherey+2);
  write('   Expanded memory frame segment     ',hex(emm_seg),'H');
  gotoxy(x,wherey+1);
  write('   Expanded memory pages in system   ',total:5);
  gotoxy(x,wherey+1);
  write('   Expanded memory pages available   ',free:5);
  gotoxy(x,wherey+1);
  write('   Expanded memory pages in use      ',used:5);
  gotoxy(x,wherey+1);
  write('   Expanded memory pages used by DEMO',pages:5);
  gotoxy(x,wherey+1);
  write('   Expanded memory handles in use    ',handles:5);
  gotoxy(x,wherey+2);
  write('Press any key to continue.... ');
  read(kbd,k);
end;

procedure handler;
begin
Inline(
   $5D                    {              pop bp)
  /$5D                    {              pop bp)
  /$FA                    {              cli)
  /$2E                    {              cs:)
  /$8C/$1E/>SAVEDS        {              mov word ptr [>saveds],ds)
  /$2E                    {              cs:)
  /$8C/$06/>SAVEES        {              mov word ptr [>savees],es)
  /$2E                    {              cs:)
  /$8C/$16/>SAVESS        {              mov word ptr [>savess],ss)
  /$2E                    {              cs:)
  /$89/$26/>SAVESP        {              mov word ptr [>savesp],sp)
  /$2E                    {              cs:)
  /$8E/$1E/>DATASEG       {              mov ds,word ptr [>dataseg])
  /$2E                    {              cs:)
  /$8E/$16/>STACKSEG      {              mov ss,word ptr [>stackseg])
  /$2E                    {              cs:)
  /$8B/$26/>STACKPTR      {              mov sp,word ptr [>stackptr])
  /$FA                    {              sti)
);
  save_cursor;
  save_screen;
  main_proc;
  restore_screen;
  restore_cursor;
Inline(
   $FA                    {              cli)
  /$2E
  /$8E/$1E/>SAVEDS        {              mov ds,word ptr[>saveds])
  /$2E                    {              cs:)
  /$8E/$06/>SAVEES        {              mov es,word ptr[>savees])
  /$2E                    {              cs:)
  /$8E/$16/>SAVESS        {              mov ss,word ptr[>savess])
  /$2E                    {              cs:)
  /$8B/$26/>SAVESP        {              mov sp,word ptr[>savesp])
  /$FB                    {              sti)
  /$CB                    {              return far)
);
end;

{All functions and procedures above this point will be moved to
EMS}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

4

```
procedure dummy;
begin
end;

var
  lastx,lasty,lastvar,xcode : integer;

procedure initialize;
var r : regpack;
begin
  r.ah := $0F;
  intr($10,r);
  color := r.al <> 7;
  r.ah := $03;
  intr($10,r);
  if color then buffer_loc := $B800 else buffer_loc := $B000;
end;

function emm_present : boolean;
var
  ems_str : array[1..8] of char;
begin
  ems_str := '        ';
  move(mem[vector[$67].segment:10],ems_str,8);
  emm_present := ems_str = 'EMMXXXX0';
end;

function emm_segment : integer;
var r : regpack;
begin
  r.ah := $41;
  intr($67,r);
  emm_segment := r.bx;
end;


function get_handle(var handle : integer) : boolean;
var r : regpack;
begin
  r.ah := $43;
  r.bx := 8;
  intr($67,r);
  if r.ah = 0 then
    begin
      handle := r.dx;
      get_handle := true;
    end
  else
    begin
      r.ah := $42;
      intr($67,r);
      handle := r.bx;
      get_handle := false;
    end;
end;

begin
  initialize;
  demo_main := ofs(handler);
  if not emm_present then
    begin
      writeln('Expanded memory driver not present - aborting!',^g,^g);
      halt;
    end
  else
    begin
      if not get_handle(emm_handle) then
        begin
          writeln;
          writeln('Insufficient memory pages available - aborting!',^g,^g);
          writeln('EM Manager reports ',emm_handle,' pages available.');
          halt;
        end;
      if emm_version <> '3.2' then
        begin
```

5

```
          writeln('Not EMS version 3.2 - found instead ',emm_version,
                  ' - aborting',^g,^g);
          halt;
        end;
      emm_seg := emm_segment;
      map_memory(0,0);
      map_memory(1,1);
      map_memory(2,2);
      map_memory(3,3);
      fillchar(mem[emm_seg:0],$FFFF,0);
      lasty := ofs(dummy);
      lastx := lasty shr 4;
      dataseg := emm_seg + lastx + 2;
      stackseg := emm_seg;
      stackptr := $BFFF;
      main_buffer := ptr(emm_seg,$C000);          (page 6)
      aux_buffer  := ptr(emm_seg,$D000);          (page 6)
      move(mem[cseg:0],mem[emm_seg:0],ofs(dummy));
      move(mem[dseg:0],mem[dataseg:0],ofs(lastvar));
    end;
end.
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;RAM resident KERNEL driver for main program in expanded memory
;Copyright 1987 by John J. Newlin, 4060-228 Rosenda Court, San
;Diego, CA 92122
;All rights reserved
;Assemble with MASM, link with LINK, convert to .COM with EXE2BIN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Equates referencing offset addresses in the main program.  Addresses
;derived from Turbo Pascal main program - see article text.
activate      equ   2D9FH
demo_main     equ   2DA6H
emm_handle    equ   2DAAH
loaded_in_emm equ   2DAEH

code segment
     assume cs:code
org 100h

begin:
      jmp initialize              ;jump directly to initialization code

demo LABEL DWORD                  ;label for far call to main prog
demo_ofs            dw 0          ;offset of main prog handler
demo_seg            dw 0          ;segment of main prog
i16 LABEL DWORD                   ;label for far call/jump to int 16h
i16_ofs             dw 0          ;offset of original int 16h
i16_seg             dw 0          ;segment of original int 16h
save_ss             dw 0          ;save caller's SS reg
save_sp             dw 0          ;save caller's SP reg
save_ax             dw 0          ;save caller's AX reg
hotkey              db 0          ;hotkey value
active              db 0          ;active flag
handle              dw 0          ;EMM handle
local_stack         dw 20 dup(0)  ;local stack area
stkptr              dw 0          ;local SP

save_map:           push ax
                    push dx
                    mov dx,cs:handle              ;DX gets our handle
                    mov ah,47h                    ;save mapping context
                    int 67h                       ;call EMM
                    pop dx
                    pop ax
                    ret

restore_map:
                    push ax
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

6

```
                push dx
                mov dx,cs:handle        ;DX gets handle
                mov ah,48h              ;to restore mapping context
               int 67h                 ;call EMM
               pop dx
               pop ax
               ret

map_memory:
                push ax
                push bx
                push cx
                push dx
                mov cx,4                ;will map 4 pages
do_again:       mov dx,cs:handle        ;DX gets handle
                mov bx,cx               ;BX gets logical page
                mov al,cl               ;AL gets physical page
                dec bx                  ;decrement logical page
                dec al                  ;decrement physical page
                mov ah,44h              ;set up for EMM call
                int 67h                 ;call EMM
                loop do_again           ;do again (3,2,1,0)
                pop ax
                pop bx
                pop cx
                pop dx
                ret

run_demo:
                call save_map           ;save mapping context
                call map_memory         ;map our pages in
                mov cs:active,1         ;set active flag
                mov cs:save_ax,ax       ;save AX reg
                cli                     ;turn off interrupts
                mov cs:save_ss,ss       ;save caller's SS reg
                mov cs:save_sp,sp       ;save caller's SP reg
                mov ax,cs               ;get our CS
                mov ss,ax               ;put in SS
                mov ax,offset cs:stkptr ;get offset of local stack
               mov sp,ax               ;put in SP
               sub sp,2                ;decrement SP
               sti                     ;enable interrupts
               push ax                 ;save
               push bx                 ; caller's
               push cx                 ;  regs
               push dx                 ;   on
               push di                 ;    our
               push si                 ;     local
               push bp                 ;      stack
               push es                 ;
               call cs:[demo]          ;call our main prog
               pop es                  ;restore
               pop bp                  ; caller's
               pop si                  ;  regs
               pop di                  ;   from
               pop dx                  ;    our
               pop cx                  ;     local
               pop bx                  ;      stack
               pop ax                  ;
               cli                     ;turn off interrupts
               mov ax,cs:save_ss       ;get caller's SS reg
               mov ss,ax               ;put in SS
               mov sp,cs:save_sp       ;restore caller's SP
               mov ax,cs:save_ax       ;restore caller's AX
               sti                     ;enable interrupts
               mov cs:active,0         ;reset active flag
               call restore_map        ;restore mapping context
               ret                     ;return

int16           proc far                ;use far attr for returns
                cmp ah,98h              ;regular int 16 request?
                jb keys16               ;yes

n00:            cmp ah,9Fh              ;request to remove?
                jne l1                  ;no
                mov ax,cs:demo_seg      ;yes, prepare to remove
```

7

```
                mov es,ax               ;ES gets frame segment
                mov ah,45h              ;prepare to release EM
                mov dx,cs:handle        ;load handle in DX
                int 67h                 ;call EMM to release
                cli                     ;turn of interrupts
                push ds                 ;save DS
                mov ax,cs:i16_seg       ;get orginal int 16h seg
                mov ds,ax               ;put in DS
                mov dx,cs:i16_ofs       ;DX = original int 16h ofs
                mov ax,2516h            ;set up to re-install
                int 21h                 ;call DOS to do it
                pop ds                  ;restore ds
                mov bx,2Ch              ;2Ch in PSP has segment
                mov ax,cs:[bx]          ;get mem block segment
                mov es,ax               ;put in ES for DOS
                mov ah,49h              ;set up to release
                int 21h                 ;call DOS to release
                mov ax,cs               ;our segment
                mov es,ax               ;put in ES for DOS
                mov ah,49h              ;set up to release
                int 21h                 ;call DOS to release
                sti                     ;enable interrupts
                mov ah,4Ch              ;set up to teminate
                int 21h                 ;call DOS to terminate

l1:
                cmp ah,9Bh              ;requ for presence flag?
                jne keys16              ;no
                mov ax,0ABCDh           ;indicator in AX
                iret                    ;return to caller

no_action16:
                jmp cs:[i16]            ;jmp to original int 16h

keys16:
                cmp cs:active,1         ;is demo active?
                je no_action16          ;yes - depart
                cmp ah,1                ;buffer status req?
                ja no_action16          ;no - depart
                je checkchar            ;yes - process

getchar:        mov ah,0                ;set up to get a char
                pushf                   ;to fake int call
                call cs:[i16]           ;call original int 16h
                cmp al,0                ;alt key pressed?
                jne exit0               ;no - depart
                cmp ah,cs:hotkey        ;our hot key?
                je action1              ;yes - process
exit0:          iret                    ;no - return to caller

action1:
                call run_demo           ;call our main prog
                jmp getchar             ;go check key buffer

checkchar:
                mov ah,1                ;set up to check buffer
                pushf                   ;to fake int call
                call cs:[i16]           ;call original int 16h
                pushf                   ;save flags
                jz exit16               ;no key in buffer - depart
                cmp al,0                ;alt key pressed?
                jne exit16              ;no - depart
                cmp ah,cs:hotkey        ;our hot key pressed?
                jne exit16              ;no - depart
                mov ah,0                ;remove key from buffer
                call cs:[i16]           ;call original int 16h
                call run_demo           ;call our main prog
                jmp checkchar           ;go check buffer

exit16:
                popf                    ;discard flags
                ret 2                   ;return to caller
int16           endp


initialize      proc
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MISOSYS

SOFTWARE FOR YOUR P.C.

Order Now 800-MISOSYS (647-6797)

ED/ASM-86™

- 80X86 Assembly Language development

An integrated editor, assembler, linker, and debugger for 8086, 80186, 80286, and 8087 assembly and disassembly. All features are in a single EXE file.

- a full-featured line EDITOR with intra-line editing, block move and copy, partial save and load.
- a full-screen turbo-type editor is supported on CGA-equipped systems.
- the ASSEMBLER supports structured coding; can directly generate a COM or EXE file; assembly to memory for faster debugging.
- The DEBUGGER includes symbolic disassembly of arbitrary code to disk.

ED/ASM-86 [M-86-030] . . $124.95 + $5 S&H

RATFOR-86™

- Rational FORTRAN translator

A professional implementation of RATional FORtran by James Beard, Ph.D. Provides structure and greater portability to FORTRAN programs to reduce your programming time and effort dramatically over that required when writing directly in FORTRAN. Includes tutorial reference manual.

- compiler has extensive macro support
- "arith" macro for binary arithmetic
- short form READ and PRINT
- enhanced switch case support
- Requires a FORTRAN compiler

RATFOR-86 [M-86-73] . . . $99.95 + $5 S&H

MISOSYS Inc.
P.O. Box 239
Sterling, VA 22170-0239
703-450-4181 MC, VISA, CHOICE
800-MISOSYS/647-6797

VA residents add 4½% sales tax,
S&H: Canada add $1;
Foreign use S&H times 3

Number **121** on the Reader Service Card


```
                mov ax,cs                       ;get code seg value
                mov ds,ax                       ;set DS to same
continue:       mov ah,9Bh                      ;set up for int 16 query
                int 16h                         ;call it
                cmp ax,0ABCDh                   ;is DEMO loaded?
                jne not_loaded                  ;no, continue
                mov dx,offset loaded_msg        ;yes, it's loaded
                mov ah,9                        ;set up to write msg
                int 21h                         ;write it
                mov ah,4Ch                      ;terminate
                int 21h
not_loaded:     mov ax,3567h                    ;set up to get int 67h seg
                int 21h                         ;call DOS
                mov di,10                       ;offset of EMM string
                mov si,offset emm_name          ;offset of our string
                mov cx,8                        ;8 chars to check
                cld                             ;set direction flag forward
                repz cmpsb                      ;compare string
                jz emm_ok                       ;compare ok?
                mov dx,offset emm_msg           ;no
                mov ah,9                        ;set up for error msg
                int 21h                         ;call DOS to write
                mov ah,4Ch                      ;set up to terminate
                int 21h                         ;terminate
emm_ok:         mov ah,41h                      ;set up to get EMM segment
                int 67h                         ;call EMM
                mov cs:demo_seg,bx              ;save it
                mov es,bx                       ;put it in ES
                mov ax,es:[loaded_in_emm]       ;get DEMO flag
                cmp ax,9876h                    ;is it there?
                je demo_present                 ;yes
                mov dx,offset not_loaded_msg    ;no
                mov ah,9                        ;set up for error msg
                int 21h                         ;call DOS to write
                mov ah,4Ch                      ;set up to terminate
                int 21h                         ;terminate
demo_present:   mov al,byte ptr es:[activate]   ;get hot key
                mov hotkey,al                   ;save it
                mov ax,word ptr es:[emm_handle] ;get EMM handle
                mov handle,ax                   ;save it
                mov ax,word ptr es:[demo_main]  ;get DEMO handler offset
                mov demo_ofs,ax                 ;save it

                mov ax,3516h                    ;set up to get int16 addr
                int 21h                         ;call DOS to get it
                mov i16_seg,es                  ;save segment
                mov i16_ofs,bx                  ;save offset
                mov ax,2516h                    ;set up to replace with ours
                mov dx,offset int16             ;offset into DX
                int 21h                         ;call DOS to install

                mov dx,offset sign_on           ;sign on msg offset
                mov ah,9                        ;set up to write
                int 21h                         ;call DOS to write

                mov dx,offset initialize        ;last byte of resident code
                int 27h                         ;terminate & stay resident

emm_name         db 'EMMXXXX0',0
emm_msg          db 'No EMM available - aborting',7,'$'
loaded_msg       db 'KERNEL in memory - aborting',7,'$'
sign_on          db 'KERNEL is loaded and DEMO is operational in EMS','$'
not_loaded_msg   db 'DEMO not in expanded memory',7,'$'

initialize      endp
code            ends
                end begin
```

RANGE ERROR.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Complete Accounting Software with Source Code

Create Custom Vertical Accounting Modules

• General Ledger • Inventory
• Accounts Payable• Cost Accounting
• Accounts Receivable

Buy all five software modules for $200 or purchase them separately for $50 each. Each module comes complete with full documentation, an executable file and documented pretty-printed source code.The modules share an easy-to-use common user interface and work together to make accounting tasks simple. A low-cost OEM license is now available for software developers.

System requirements: Turbo® Pascal 3.0 or above, the Turbo Pascal Toolbox, and an IBM® PC or compatible. Each module is shipped on a DSDD 5 1/4" diskette. Shipping charges: $5 US, $10 Canada. Send check or money order to:

PS BUSINESS SYSTEMS

PS Business Systems
4400 Wallingford Ave N.
Suite 3
Seattle, WA 98103
(206) 632-4568

Number **82** on the Reader Service Card

DISK FORMAT CONVERSION

XenoCopy-PC

PC-DOS program lets your PC

$79.95 + $5.00 S/H
Sales Tax if CA.
VISA MasterCard UPS COD

READ / WRITE /
FORMAT / DUPLICATE
Disks from over 300 other micros

Upgrades available from all previous versions for only $25.00 Call for Authorization

To Order Contact:

XENOSOFT™

1454 Sixth Street, Berkeley, CA 94710
(415) 525-3113

this was on purpose.

good software is it's own reward.

Number **123** on the Reader Service Card

Software Tools

For Programmers & Non-Programmers

Get 'State of the Art' performance and save valuable time with these high quality utilities!

Opt-Tech Sort™

Opt-Tech Sort is a high performance Sort/Merge/Select utility. It can read, sort and write a file faster than most programs can even read the data. Example: 1,000 records of 80 bytes can be read, sorted and a new file written in less than 10 seconds (IBM XT). Opt-Tech Sort can be used as a stand-alone program or called as a subroutine to over 25 different programming languages.

All the sorting, record selection and reformatting facilities you need are included. A partial list of features includes: The ability to process files of any size. Numerous filetypes are supported including Sequential, Random, Delimited, Btrieve, dBASE II & III and many others. Up to 10 key fields can be specified (ascending or descending order). Over 16 different types of data supported. Powerful record selection capability allows you to specify which records are to be included on your output. Record reformatting allows you to change the structure of your output record and to output special fields such as record numbers for use as indexes.

MS-DOS $149. ★ NEW ★ Xenix $249.

★ NEW ★ VERSION **On-Line Help™** ★ NEW ★ VERSION

On-Line Help allows you to easily add "Help Windows" to all your programs. On-Line Help is actually two help packages in one. You get BOTH Resident (pop-up) and Callable Help Systems.

The resident version allows you to add help to any system. Your Help System is activated when the "Hot Keys" that you specify are pressed. You can then chain between help windows in any manner you desire.

The callable version allows you to easily display help windows from your programs. A simple call to the help system makes the window appear. The original screen is automatically restored when the help window is cleared. On-Line Help is callable from over 20 different languages.

You have full control over the help window content, size, color and location.

MS-DOS $149. **Demo $10.** (apply toward purchase).

Scroll & Recall™

Scroll & Recall is a resident screen and keyboard enhancement. It allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations). Also allows you to easily recall and edit your previously entered DOS commands without retyping.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it. MS-DOS $69.

Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted.

To order or to receive additional information just call and receive immediate highly qualified attention!

Opt-Tech Data Processing
P.O. Box 678 — Zephyr Cove, NV 89448
(702) 588-3737

Number **16** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Note on Optimizing Turbo Pascal

by
Jon Greenblatt

Thanks to Bela Lubkin for the Turbo Pascal compiler patch (*PJ*, Volume 5.3, May/June 1987); however, Turbo needs those E90000s as a target for jumps when you use the NULL statement. For example,

```
procedure clio;
  begin
    if IOresult <> 0 then { arf }
  end;
```

will crash heavily (run-time) under the patched compiler, and

```
  case c of
    'T': go_forward;
    'F': go_back
    '?': { stand still }
  end;
```

will put the compiler into a tizzy (compile-time). Easy to fix: Replace every NULL statement by a nop: INLINE($90).

Here are some tips in return:

- This finesse lets you skip over the prolog code entirely on parameterless procedures:

```
procedure PROCname;
begin
  { do some biz with global variables, presumably inline }
  inline($C3) { ret before compiler generated cleanup }
end;

inline($E8/>$FFFF-*+PROCname-1+prolog); {instead of: PROCname;}
```

And if you're not timid, you can use it for parameterized procedures, too. For example,

```
  { save_used_marker(Tlist); }
  fnPTR:=Addr(Tlist); }
  inline($E8/>$FFFF-*+save_used_marker-1+prolog);
```

where fnPTR is a global generic pointer "buffer," and the procedure code is modified to access fnPTR instead of the stack (parameter).

Normally, prolog = 7; using Bela's patch, prolog = 4.

- The following is quicker than the normal while p<>nil do—if p is a simple global/local pointer (not a record field or array reference)

```
while ((Seg(p^) or Ofs(p^))<>0) do
  begin
    { something useful }
    p:=p^.next
  end;
```

since it generates straight code, not a call to the Turbo library, despite the extra memory reference to p.

It's even faster to create a typed NIL pointer by choosing some invalid value (best defined as an untyped constant) for a scalar or integer field of p^. Set up a sentry (sentry^.f:=invalid), assign it instead of NIL, and use a loop condition:

```
  while p^.f<>invalid do
    begin
      { something useful }
      p:=p^.next
    end;
```

-
```
type regpack=record
               ax,bx,cx,dx,bp,si,di,ds,es,flags: integ
             end;
var result: regpack;
```

requires referencing record fields for setup and DOS function retrieval (after MsDOS call), not to mention byte shifts to place parameters in the proper registers.

```
var
  ALreg, AHreg, BLreg, BHreg, CLreg, CHreg,
    DLreg, DHreg: byte;
  BPreg, SIreg, DIreg, DSreg, ESreg,
    FLAGSreg: integer;
  AXreg: integer absolute ALreg;
  BXreg: integer absolute BLreg;
  CXreg: integer absolute CLreg;
  DXreg: integer absolute DLreg;
```

allows references directly to memory locations, using MsDOS(ALreg).

- String ops do library calls:

Shun

```
s:='';
```

It takes 24 times as long as

```
s[0]:=#0;
```

If the two strings being compared are usually not the same (and have varying lengths), compare lengths first:

```
function index(var s: s15): byte;
var i: integer;
    L: char absolute s;
begin
  index:=255;
  for i:=1 to list_top do
    if list[i]^.name[0] = L then
      if list[i]^.name = s then
        begin
          index:=i;
          EXIT
        end
end;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Here, the extra array reference and pointer/rec field access look verbose but have insignificant cost.

- Note that WITHs are expensive. For every open WITH, there should be at least two references into the record on average; otherwise, use a chain of accesses to get at individual fields. "On average" means dynamically—not how many times a field is mentioned, but how many times it is used. As this depends on how often conditions are true, etc., experimentation is key here.

```
{ fast, ugly code: }
if pp^.trans^.tt^.side_effects then { this is rarely true }
  with pp^.trans^.tt^ do              { not "with pp^, trans^.tt^" }
    if cp[j]<>nil then
      if restrix_check(cp[j],stack[stack_top], pp^.trans)
        then { aff } INLINE($90);    { not "trans" }
```

- In Turbo (despite what they teach you in compiler courses), CASE statements are not jump tables. They're a sequence of IF–THENs, so the most heavily trafficked branches should come earliest:

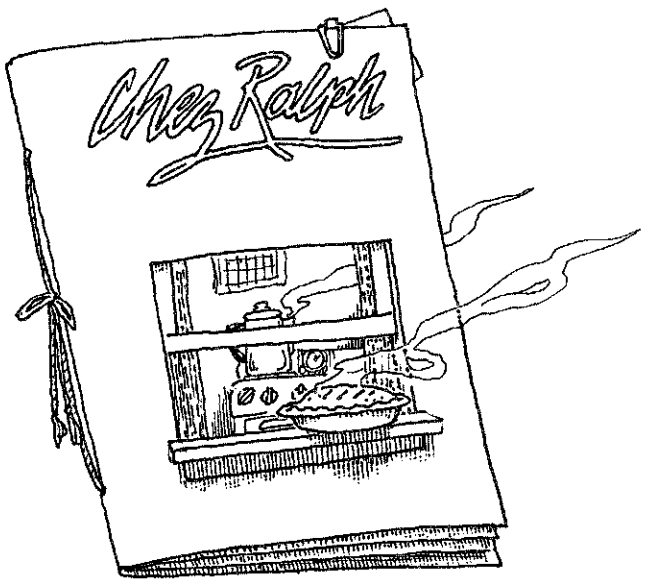
```
case b of
       5: c:='A';
  1..10: c:='B'
end;
```

Looks dumb, runs smart if b were 5 most of the time. There'd usually be one compare; otherwise, three (b = 5?, b > = 1?, and b < = 10?).

Needless to say, some of this stuff is for squeezing cycles from a stone, not for the prototyping stage. □

Jon Greenblatt is president of ELFSOFT, makers of natural language frontend software. He can be contacted at

210 West 101 Street
New York City, NY 10025



MEND WITH WINDOWS

Introducing

Quaid Analyzer

the tool that created CopyWrite

Now you can debug your own programs with a professional quality debugger - the one that unraveled every form of copy-protection used on the PC.

With the Quaid Analyzer, you can:

- ☐ See occurences of any interupt, with its meaning shown on the screen.
- ☐ View memory as text or instructions, scrolling as easily as you do with an editor.
- ☐ Run until a memory location or I/O port is changed.
- ☐ Protect your hard disk from accidental destruction.
- ☐ Analyze software without the source, even when it uses countermeasures to thwart tracing.
- ☐ See all stages of the boot load.

We kept the Quaid Analyzer off the market to avoid helping publishers with copy-protection. Now that copy-protection is gone, we can sell it to you.

The Quaid Analyzer is a software tool occupying 100K bytes. It runs on any IBM PC and most MS-DOS systems without hardware modification.

To order Call **(416) 961-8243**

Quaid Analyzer $99 U.S.

All orders shipped at our expense within a day. All major credit cards accepted.

or return coupon to:
45 Charles St. East
Third Floor, Dept. 603
Toronto, Ontario. M4Y 1S2

Payment method MC-Visa-Amex-Diners-Check
Card No. _____
Expiry Date _____
Name _____
Address _____
City/State _____
Phone No. _____
Signature _____

Quaid Software Limited

Ask about Disk Explorer the program that takes over where Quaid Analyzer leaves off.

Contact this Advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Proposed ANSI C Language Standard

by
Rex Jaeschke

© *Rex Jaeschke, 1987. All rights reserved.*

Throughout this article the compilers Microsoft C V4, Lattice C V3, and Wizard C V3, are referred to as MSC, LATC, and WIZC, respectively.

Numerical Limits

The Standard specifies a number of macros that define certain numerical characteristics of the target environment. All conforming implementations must contain these definitions.

The integral characteristics are defined in the header **limits.h**, and the floating-point characteristics are in **float.h**.

limits.h

While the exact value of each macro is implementation-defined, it must be equal to or greater in magnitude (absolute value) than that provided for by the Standard, with the same sign. The names and purposes of the macros are as follows:

CHAR_BIT	8	number of bits in a byte
SCHAR_MIN	-127	minimum value for signed char
SCHAR_MAX	+127	maximum value for signed char
UCHAR_MAX	255U	maximum value for unsigned char
CHAR_MIN	**	minimum value for char
CHAR_MAX	**	maximum value for char
SHRT_MIN	-32767	minimum value for short int
SHRT_MAX	+32767	maximum value for short int
USHRT_MAX	65535U	maximum value for unsigned short int
INT_MIN	-32767	minimum value for int
INT_MAX	+32767	maximum value for int
UINT_MAX	65535U	maximum value for unsigned int
LONG_MIN	-2147483647	minimum value for long int
LONG_MAX	+2147483647	maximum value for long int
ULONG_MAX	4294967295U	maximum value for unsigned long int

This means that in order to conform, an **int** must be at least 16 bits and a **long int** must be at least 32 bits. Note the use of the U suffix to designate an unsigned integer constant. This may be combined with **L** (or **l**) to give **UL**, **LU**, **ul**, **lu**, **Ul**, or **uL**—all of which mean the same thing, **unsigned long int**. The new unary plus operator is also used in the examples above.

On systems that don't yet support the U suffix, the maximum value for unsigned integer types can be specified as a hexadecimal constant as follows:

USHRT_MAX	0xffff	unsigned int	16 bits
UINT_MAX	0xffff	unsigned int	16 bits
ULONG_MAX	0xffffffff	unsigned long int	32 bits

Note the subtle difference between these and the following:

USHRT_MAX	65535	signed long int	32 bits
UINT_MAX	65535	signed long int	32 bits
ULONG_MAX	4294967295	unsigned long int	32 bits

According to the Standard, the type of an integer constant that has no suffix and is a decimal value is the first in the following list in which its value can be represented. The list is: **int**, **long int**, and **unsigned long int**. By this rule, then, the three constants above have types **long int**, **long int**, and **unsigned long int**, respectively, whereas their hexadecimal counterparts have type **unsigned int**, **unsigned int**, and **unsigned long int**, respectively. For the most part this difference is academic; however, a function call of the form **f(USHRT_MAX);** will cause problems if 32 bits are pushed on the stack yet only 16 bits are expected.

UCHAR_MAX is a special case since 255 (and 0xff) has type **signed int** whereas **255U** has type **unsigned int**. While the two are not the same type, they do have the same size on DOS and don't cause problems when used as function arguments.

The minimum value of a signed object has the same magnitude as its positive counterpart to allow for ones-complement implementations. Compilers running on twos-complement systems, such as the Intel CPUs, will typically have negative values one fewer than those specified above; for example,

SCHAR_MIN	-128
SHRT_MIN	-32768

MSC does not do this, however. Those shown above are all of MSC's signed minimums. This can lead to problems when porting code to MSC since on some other twos-complement machine **SCHAR_MIN** will be -128 whereas on MSC it is -127. Since -127 is *not* the minimum value for a signed **char** (-128 is), the value is also misleading. Beware when conditionally compiling code based on these macros on MSC.

The values for **CHAR_MIN** and **CHAR_MAX** depend on whether or not plain chars are signed. If they are, these macros are the same as **SCHAR_MIN** and **SCHAR_MAX**. If plain chars are unsigned, **CHAR_MIN** is 0 and **CHAR_MAX** is the same as **UCHAR_MAX**. In MSC, plain chars are signed by default. You can change them to unsigned by using the compile-time switch /J. Note, though, that the header contains the following definitions:

CHAR_MAX	127
CHAR_MIN	0

These limits correspond to neither signed nor unsigned chars—they define the range of 7-bit positive values only. So while these macros are defined, they are not strictly correct.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

In theory you should be able to portably detect whether or not a char is signed by testing **CHAR_MIN** against 0. If it tests true, the char is unsigned. This will work with MSC; however, the range of values as calculated by **CHAR_MAX – CHAR_MIN** is still only 127, not 255.

Many compilers support the use of the **sizeof** operator in preprocessor arithmetic since their preprocessors are actually part of the compilers proper; ANSI C, however, does not allow **sizeof** in preprocessor evaluated expressions and thus outlaws the following construct:

```
#if sizeof(int) == 2
long total;
#else
int total;
#endif
```

Conditionally compiling based on machine word-size is common. Here, the example assumes that if it isn't running on a 16-bit system it's on a 32-bit machine. To achieve the same result, you must now use something like

```
#include <limits.h>

#if INT_MAX < LONG_MAX
long total;
#else
int total;
#endif
```

float.h

This header has much the same purpose as **limits.h** except that it defines the floating-point characteristics of the target system. The meaning and formulas applied to each of these can be complex, and you are referred to your compiler manual or a copy of the draft Standard for specific details.

FLT_RADIX	2
FLT_ROUNDS	
FLT_MANT_DIG	
DBL_MANT_DIG	
LDBL_MANT_DIG	
FLT_EPSILON	1E–5
DBL_EPSILON	1E–9
LDBL_EPSILON	1E–9
FLT_DIG	6
DBL_DIG	10
LDBL_DIG	10
FLT_MIN_EXP	
DBL_MIN_EXP	
LDBL_MIN_EXP	
FLT_MIN	1E–37
DBL_MIN	1E–37
LDBL_MIN	1E–37
FLT_MIN_10_EXP	–37
DBL_MIN_10_EXP	–37
LDBL_MIN_10_EXP	–37
FLT_MAX_EXP	
DBL_MAX_EXP	
LDBL_MAX_EXP	
FLT_MAX	1E+37
DBL_MAX	1E+37
LDBL_MAX	1E+37
FLT_MAX_10_EXP	+37
DBL_MAX_10_EXP	+37
LDBL_MAX_10_EXP	+37

ANSI C contains a lot of support for the numerical community who have previously shunned C because of its lack of support for the types of arithmetic they need. To make C more attractive to this large user community, ANSI C has done the following:

- Specified the header **float.h**.
- Added the new type **long double**.
- Allowed floating-point arithmetic to be performed in any precision provided the result is the same as if the widening rule (float to double) were followed.
- Allowed float values not to be widened when used as function arguments.
- Allowed floating-point constants to have type **double** (as before), type **float** by appending a suffix of **F** or **f**, or **long double** by appending a suffix of **L** or **l**.

MSC supports a number of these macros, but they are defined in **limits.h**, *not* in **float.h**. Also, MSC has several macros beginning with the prefix **LDBL_** indicating the new **long double** data type; however, the compiler does not support that type. The values for these macros are the same as those for their double counterparts, indicating that for MSC, the type **long double** will be the same as **double**.

LATC has a header called **limits.h**, but it contains only one entry, the definition for a macro **HUGE_VAL** used in floating-point computations in the math library. ANSI C has this defined in **math.h**.

The WIZC version of **limits.h** has all the ANSI macros, and it has the correct values for the negative numbers except for **INT_MIN**, which is set to –32767 rather than –32768. I suspect that this is an oversight. One problem, though, is that these minima are defined as decimal numbers using something like

```
#define SHRT_MIN  -32768
```

The type of this constant is **signed long int**, *not* **signed int** as expected, because C *does not support negative constants*. The term –32768 is an expression made up of the unary operator and the positive number 32768, and since 32768 has type **long int** (it won't fit into a 16-bit int), negating it makes it a **long int** with a negative value.

On systems that don't yet support the U suffix, the maximum value for unsigned integer types can be specified as a hexadecimal constant.

WIZC also has a skeletal **float.h** similar to MSC's. One major difference, though, is that WIZC supports the new type

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

long double, in both declarations and floating-point constants.

Future Meetings

The schedule for future ANSI meetings is: September 14–18, Framingham, Massachusetts; January 25–29, 1988, at Austin, Texas. □

Rex Jaeschke is an independent computer consultant, writer, and lecturer. He is also a member of the X3J11 C Standards Committee, Co-founder and Editor of The C Journal, *and the C language Editor of* DEC PROFESSIONAL. *If you would like to comment on or question any aspect of the Proposed ANSI C Standard, write to:*

Rex Jaeschke
2051 Swans Neck Way
Reston, VA 22091

Rex will address your concerns in future columns as space permits.

"If you need source code, make sure your wallet is wide open or get VITAMIN C."
- Computer Language, June, 1987

Vitamin C

It's good for your system!

- Professional C function library
- 30 day money back guarantee
- Multiple bullet proof windows
- Easy full screen data entry
- Unlimited data validation
- Context sensitive help manager
- Menus like Lotus and Mac
- Programmable keyboard handler
- Text editor routines
- No royalties or runtime fees
- Library source included FREE
- Free technical support
- Free BBS at (214)418-0059
- Supports all major compilers including Microsoft 5.0
- VCScreen code generator too!
- UNIX version avaialable

Fast, flexible, versatile, reliable. Vitamin C delivers the vital combination that software professionals demand. Highly efficient, professionally crafted C code provides lightning fast displays required by today's window intensive programs.

High level functions provide maximum productivity and require little supporting code. Plus, Vitamin C's versatile, open ended design is full of hooks so you can intercept and plug-in special handlers to customize or add features to most routines.

VCScreen, our screen painter / code generator speeds your development even more! Simply draw your input forms using our interactive design editor and generate perfect C source code ready to compile and link with the Vitamin C library.

Vitamin C $225
Includes all source code FREE! For IBM PC, XT, AT, PS/2 and true compatibles. Specify compiler when ordering.

VCScreen $99.95
For IBM PC, XT, AT, PS/2 and true compatibles. Requires Vitamin C.

ORDER NOW!

We ship UPS. Please include $3 for ground, $6 for 2-day air, $20 for overnight, or $30 if outside the U.S. Texas residents add 7 1/4 % sales tax. All funds MUST be in U.S. dollars drawn on a U.S. bank. Visa & MasterCard accepted.

(214)416-6447

creative PROGRAMMING
Box 112097 • Carrollton, Texas 75011

Windows • Data Entry • Menus • Help • Text Editing
Plus... All Source Code FREE!

Contact this Advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MICROCACHE™ and SPEEDPRINT™

The hottest PC utilities you can get.
Or your money back!

MICROCACHE™ monitors disk use and "learns" to anticipate your data accesses. Then, by satisfying these requests directly from RAM instead of disk, your programs run as if they are SUPERCHARGED.

- 100% Transparent Operation
- Uses any spare RAM available
- EMS/EEMS memory support
- SPEEDPRINT™ buffer option
- Requires NO program changes
- For IBM and compatibles

Introductory Price $89.00

MasterCard VISA

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085

(800) 227-0644 . . . (816) 776-2700

Number **51** on the Reader Service Card

C Programmer's Toolbox

Productivity Enhancement Tools

CFlow	Generate a map of your program's subroutine hierarchy. Supports an unlimited number of program source listings.
CPrint	Beautify your program listings, making them consistant and easy to read.
PMon	Monitor your program execution and system calls, determining where the most time sensitive/critical sections are. Does not require any modification to an existing program.
*****	And more.... All from one source....

$79.95 30 day money back guarantee
CA residents add applicable sales tax

MMC AD Systems
Box 360845
Milpitas, California 95035
(408)-263-0781

Number **94** on the Reader Service Card

386•386•386•

A Note on Printer Output from Turbo C Programs

by
Bernard H. Robinson, Jr.

None of the material in this note will be news to experienced C programmers; it is intended for readers accustomed to programming in some other high level language, especially BASIC, who are attempting C for the first time with Borland International's impressive Turbo C Package. Borland wrote good manuals for Turbo C, but they should have given at least a hint for beginners about output to the printer. Books on C for beginners frequently omit discussion of printer output, too. I can remember my frustration a couple years ago when I started to learn C programming with another C compiler package.

To Borland's credit, they do show one way to do printer output, in the source code to their spreadsheet program, "MCALC," but the printer output is a few lines in many, many pages of code.

New users of Turbo C looking for a function specifically for printer output, similar to BASIC's LPRINT statement, will not find it in the library. Never fear—some of the library functions supplied for file output will also output to the printer. The accompanying program listing OUTDEMO.C shows two basic ways to address the printer and demonstrates a few other related points as well. Turbo C, like C compilers in general, incorporates two different families of functions for handling input and output, one family involving file handles and the other involving pointers to a special structure of type FILE, which is defined in the "stdio.h" include file. The functions using file handles are called "unbuffered I/O functions" or "Level 1 I/O functions." The functions using FILE pointers are called "buffered I/O functions," "Level 2 I/O functions," or "stream functions."

Output with the Level 1 I/O Functions

C I/O functions will interact with both disk files and devices such as the console and printer. I will refer to the item being opened, written to, etc., as a "channel" to avoid having to write "file or device" repeatedly. Before DOS starts to execute a program, it opens five standard input and/or output channels for Level 1 access. These are mentioned on page 15 of Appendix D of the IBM PC DOS 2.0 manual. For more details see a good reference book on DOS such as Ray Duncan's *Advanced MS DOS* published by Microsoft Press, 1986. These preopened channels are:

Handle	Channel
0	Standard input device; can be redirected, normally the console keyboard
1	Standard output device; can be redirected, normally the display screen
2	Standard error output device; cannot be redirected, normally the display screen
3	Standard auxiliary device, cannot be redirected, normally COM1:
4	Standard printer device, cannot be redirected, normally LPT1:

DOS assigns an integer number, called a file "handle," to each channel as shown above. The C functions used to open I/O channels for Level 1 access use DOS function calls that return the file handles that are in turn returned by the C functions. One group of C functions require these file handles to specify the channel they are writing to or reading from. These C functions can use the handles listed above without having to open these standard channels explicitly because DOS has opened them already when your program starts to execute. OUTDEMO.C illustrates the use of the Turbo C "write" function to output a string to the printer using handle no. 4.

Output with the Level 2 I/O Functions

As mentioned above, the second general method that C provides for I/O involves a structure of type FILE defined in the "stdio.h" include file. FILE, in capitals, may be considered a user-defined data type identifying this type of structure. Although you will find a complete description of the FILE structure in "stdio.h," it really is not necessary to understand the structure for most uses of it. When a C function opens a channel for Level 2 access, a block of memory is set up for a FILE structure for the channel and the function returns a pointer to this block of memory. We will call the family of functions using these FILE pointers "Level 2 I/O functions" although they are also known as "buffered I/O functions" or "stream functions." It's easy to identify most functions belonging to this family when looking at declarations in the Turbo C header file listings. Level 2 I/O functions will have "FILE *fp"

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

as an argument or "FILE *" in the return declaration. The "Usage" line of the function definitions in the *Turbo C Reference Guide* usually says "FILE *stream" instead of "FILE *fp," but it means the same thing. For historical reasons involving Unix, channels opened using the FILE type functions are often referred to as a "stream" and the description of these functions in the *Turbo C Reference Guide* often speaks of reading from or writing to a "stream."

C has many more functions, and more sophisticated functions, in the Level 2 I/O family than in the Level 1 family. When you are writing to the printer, it's usually more convenient to use the Level 2 family of functions. For opening a file, C functions for both Level 1 or Level 2 access require a string parameter containing the name of the file to open. Instead of a filename, the functions also accept "PRN," "CON," and "AUX" to open channels to the printer, console, and auxiliary device, respectively. "PRN" can be opened only for output. Thus we could open the printer for Level 2 output by specifying "PRN" as the channel to open in an "open" function; however, there is also another way.

Three of the channels preopened by DOS as Level 1 functions, as was mentioned above, are also made available by Turbo C for Level 2 access. The Turbo C stdio.h file defines the macro "stdin" to mean a FILE pointer to the standard input device; thus, if "stdin" appears as the FILE pointer in an "fscanf" statement, for example, the statement will accept input from the standard input device, normally the console keyboard. So "stdin" is the FILE pointer equivalent of file handle 0. Likewise, "stdout" corresponds to file handle 1, and "stderr" to file handle 2.

In some other C compilers, two more FILE pointers, "stdaux" for file handle 3 and "stdprt" for file handle 4, are provided. OUTDEMO.C shows how to generate a FILE pointer named "stdprt" that can be used as "stdprt" is in these other compilers. The "fdopen" function returns a FILE pointer for a channel that has already been opened for Level 1 access so that Level 2 functions can be used with the channel. OUTDEMO.C uses "fdopen" to store a pointer to the printer FILE structure into the "stdprt" variable, then uses the "fprintf" function to write a string to the printer. OUTDEMO.C illustrates the traditional C language way to program a function whose return value needs to be tested. This method of placing the "fdopen" function in an "if" statement assigns the return value of the function to the pointer variable "stdprt" and then checks to see whether the return value was a NULL; if so, the program prints an error message and exits. NULL, by the way, is a macro defined in "stdio.h" in such a way that the definition changes depending upon the model (tiny, small, compact, large, etc.) being used in the compilation.

Since "stdprt" was declared outside any function, it is a global variable that can be accessed from any function in the program. If access is desired from a module of the program that is compiled separately, it must be declared as an external variable in that module.

Programs are sometimes written to send output only to the display screen, relying on redirection at the DOS command level to direct the output to the printer or a file when required. Since output redirection affects only the output sent to the standard output device ("stdout"), the programmer can use "stderr" in a function such as "fprintf" to send output to the screen that he or she does not wish to be redirected. This is useful for prompts for user input, etc., that would not look good in the redirected output.

High Level and Low Level File Access

The Level 2 I/O functions are further removed from the operating system function calls than the Level 1 functions and thus are more portable from one operating system to another, as a rule. The Level 2 functions often use Level 1 functions internally to access the channels.

We mentioned above using the "fdopen" function to attach a FILE pointer to a file already opened with a Level 1 function. There is also a function "fileno" that returns the file handle for a file already open for Level 2 access. Be very careful with these functions. One of the most useful C programming reference books in my library, *C Wizard's Programming Reference* by W. David Schwaderer (Wiley Press, 1985), warns that mixing high level and low level calls on a single open file can confuse the pointers to the current position in the file with disastrous results.

This discussion is not intended to be a tutorial on I/O for disk files. The easiest to understand discussion of C file programming for beginners that I know of is in William James Hunt's *The C Toolbox* (Addison Wesley, 1985). Unlike many books on C, this one is specifically for the IBM PC DOS operating system, rather than the Unix operating system. *The C Toolbox* discusses when to use buffered I/O (Level 2 access) and when to use unbuffered I/O (Level 1 access). It is also an easy to follow introduction to C programming in general.

Don't let the need of a FILE structure by Level 2 I/O functions cause you to confuse them with the old DOS BIOS function calls that required a file control block. There is no relation. Both families of C functions discussed above allow disk and path specifiers on the filename, but the old DOS file control block function calls did not. I don't know why the *Turbo C Reference Guide* uses "filename" as a dummy file specifier for the "fopen" function and "pathname" as a dummy name for that in the "open" function. Both can take disk and path specifiers in the filename parameter.

Don't let the need of a FILE structure by Level 2 I/O functions cause you to confuse them with the old DOS BIOS function calls that required a file control block.

Other Functions for Level 1 and Level 2 I/O

It is often useful to know which input/output functions belong to the Level 1 family and which belong to the Level 2 family. Tables 1 and 2 list the functions belonging to the two families. I am not sure that these tables are complete, but they should be nearly so. Some functions are implemented in the compiler as macros rather than as true functions. These are identified in the figures with "(m)" following the function name.

Continued...

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

__close	fdopen	open
close	filelength	__read
__create	fileno (m)	read
create	fstat	setftime
creatnew	getftime	setmode&
creattemp	ioctl	tell
dup	lock	unlock
dup2	lseek	__write
eof	__open	write

Table 1. Level 1 Functions and Macros

clearerr	fopen	getw
fclose	fprintf	putc (m)
fcloseall	fputc	putw
fdopen	fputs	setbuf
feof (m)	fread	setvbuf
ferror (m)	freopen	ungetc
fflush	fscanf	va__arg (m)
fgetc	fseek	va__end (m)
fgets	ftell	va__start (m)
fileno (m)	fwrite	vfprintf
flushall	getc (m)	vfscanf

Table 2. Level 2 Functions and Macros

Table 3 lists functions that do not require file handles or FILE pointers as parameters, but that work directly with "stdin" or "stdout." Their input and output can be redirected.

STDIN	STDOUT
cgets	cputs
cscanf	cprintf
fgetchar	printf
getchar (m)	putch
getche	putchar (m)
gets	puts
scanf	vprintf
vscanf	ungetch

Table 3. Functions and Macros for STDIN and STDOUT

I have included in the Level 2 family some functions that do not actually use a FILE pointer but operate on channels opened by this family of functions. "fcloseall" and "flushall" belong in this category. The *Turbo C Reference Guide* says that "fcloseall" closes all open "streams" except "stdin" and "stdout"; however, it does not appear to close "stdprt" when opened by "fdopen" as in OUTDEMO.C. Conversely, the documentation for Lattice C, Version 3.0, says that "fcloseall" closes all open streams, including "stdin" and "stdout," so apparently there is a difference between these two compilers in the implementation of the function. □

Bernard H. Robinson, Jr., retired four years ago from a large aerospace company, where he was responsible for both hardware and software design for computer-controlled stations that tested company products. Since retiring, he has devoted his time to writing (he is a frequent contributor to PJ*) and consulting.*

Code follows.

Periscope™ Power... Made to Order!

The unquestioned price/performance leader in PC debuggers offers you a full line of debugging systems, from software-only Periscope II-X to the full-fledged hardware breakpoint Periscope III.

Periscope debugging power is available in four models... Start with the model that fits your current needs. When you need more horsepower, upgrade for the difference in price plus $10!

When you move to another Periscope model, don't worry about having a lot more to learn... Even when you move to the most powerful model, Periscope III, an extra dozen commands are all that's involved.

A Periscope I user who recently began using Periscope III writes, *"I like the fact that within the first half hour of use I was debugging my program instead of learning to use the debugger."*

Periscope's software is solid, comprehensive, and flexible. It helps you debug just about any kind of program you can write... thoroughly and efficiently.

Periscope requires an IBM PC, XT, AT, or close compatible (Periscope III requires hardware as well as software compatibility); DOS 2.0 or later; 64K available memory; one disk drive; an 80-column monitor.

Call us with your questions; we'll be happy to send you FREE information or help you decide on the model that best fits your needs.

Periscope I includes a half-length board with 56K of write-protected RAM; break-out switch; software and manual for $345.

Periscope II includes break-out switch; software and manual for $175.

Periscope II-X includes software and manual (no hardware) for $145.

Periscope III includes a full-length board with 64K of write-protected RAM, hardware breakpoints and real-time trace buffer; break-out switch; software and manual. Periscope III for machines running up to 8 MHz is $995; for machines running up to 10 MHz, $1095.

Order Your Periscope, Toll-Free, Today!

800/722-7006

The PERISCOPE Company, Inc.

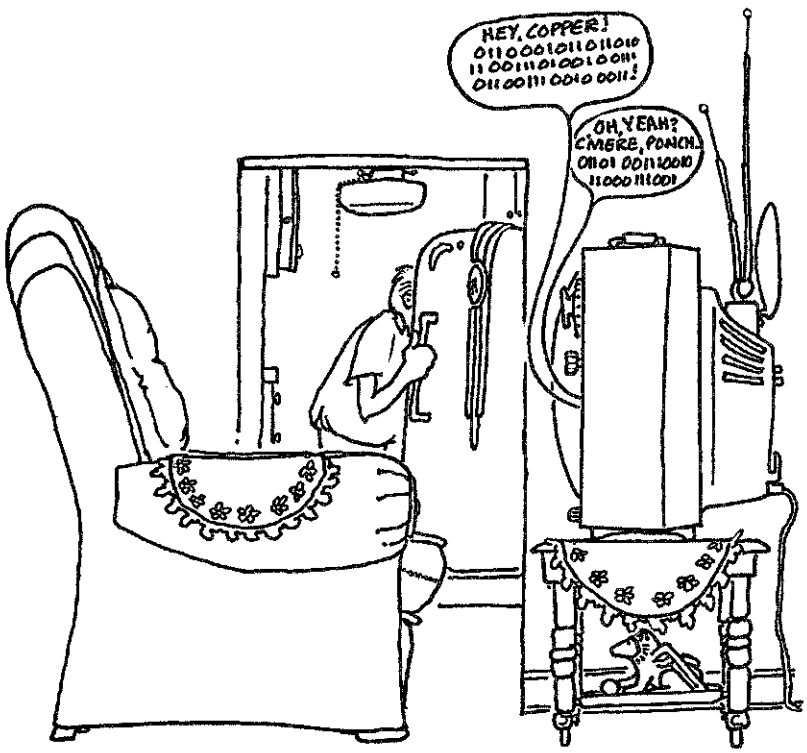
14 Bonnie Lane • Atlanta, GA 30328 • 404/256-3860

Number **6** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Code from "Printer Output" by Bernard Robinson — 1

```
/*******************************
*       name: outdemo.c
*    purpose: Demonstrate two ways of printer output for Turbo C
* written by: Bernard H. Robinson, Jr.
*       date: June 1, 1987
*    version: 005
*******************************/

#include "stdio.h"   /* For FILE structure: fdopen, fprintf, printf,
                        and  fcloseall functions */
#include "process.h" /* For exit function */
#include "io.h"      /* for write function */

FILE *stdprt;   /* declare stdprt pointer to FILE structure  */

void main()
{
    int i;
    char *s1,*s2;
    s1 = "This is a test line for the write function.\n";
    s2 = "This is a test line for the fprintf function.\n";

    write(4,s1,strlen(s1));                /* Level one printer output */

    /* Now lets go on to level two printer output */

    /* First point "stdprt" to the printer */

    if( (stdprt = fdopen( 4, "wt" ))  == NULL) {
        printf("Unable to open printer");
        i = -1;
        exit(i);
    }
    /* Now use "stdprt" to output to printer */

    fprintf(stdprt,"%s",s2);
}
/************* End of program *****************/
```



AH, THOSE OLD CHiPs RERUNS!

C PROGRAMMER!

10 Important Reasons to Use db_VISTA™ for File Management

1. **db_VISTA is written in C.**
2. **It's fast,** combining B-tree indexing with "network" or direct "set" relationships between records.
3. **It's flexible.** Use **db_VISTA** as a file manager or a complex database; single-user PC to multi-user VAX with millions of records. Upgrade easily with full compatibility!
4. **It's portable.** MS-DOS, UNIX, VMS . . . see list below.
5. **It uses space efficiently.** Non-RAM resident, only operative functions are in your run-time program.

And there's more . . .

6. **Royalty-free run-time.**
7. **Complete source code available.**
8. **SQL-based db_QUERY!** Add fast, C-linkable db_QUERY for ad-hoc queries and reports. Build an end-user interface; provide a relational view of the database.
9. **FREE tech support hotline!** 60 days free; for product or application development help. Extended support, consulting, training classes — all available.
10. **30 Day Money-Back Guarantee.** Full refund if not completely satisfied.

That's just the beginning!
Want more details? Call today!

Order Now. It's easy — simply call toll-free. We'll answer your technical questions and get you started . . . fast delivery.

Call Toll-Free Today!
1 (800) db-RAIMA
(that's 1-800-327-2462)
— OR Call 1-206-828-4636

db_VISTA™ Royalty Free Prices

	Object	w/Source
Single-User	$ 195	$ 495
Multi-User	$ 495	$ 990
VAX Multi-User	$ 990	$1,980

db_QUERY prices are the same as above

MasterCard VISA

- **Operating systems:** MS-DOS, UNIX, XENIX, ULTRIX, VMS; more . . .
- **Compilers:** Microsoft, Lattice, IBM, Computer Innovations, Aztec, Turbo C, UNIX, XENIX; more to come . . .

also works with most C libraries!

RAIMA™ CORPORATION

3055 - 112th N.E., Bellevue, WA 98004 USA
(206) 828-4636 Telex: 6503018237 MCI UW

Contact this Advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Matchup of Turbo Basic 1.0 and QuickBASIC 3.0

by
Paul P. Dymerski

The Fight of the Century

Yes, folks, the fight of the century has begun! In this corner we have Microsoft QuickBASIC Version 3.0 with 8087 support, user and standalone libraries, and the world's fastest BASIC Compiler. In the other corner we have Borland Turbo Basic Version 1.0. with 8087 support, user INCLUDE libraries, and the world's fastest BASIC Compiler.

I know you've recently seen all the hype on these two BASIC compilers. I've already seen several reviews as well. From the programmer's point of view, however, no one has yet talked about the nitty gritty features of these two compilers. No, I'm not talking about which editor displays the most colors (which seems to be important to some reviewers); I'm talking about the differences these two packages mean to your code.

In this article, we will review the features and "peculiarities" I have found so far in both the QuickBASIC 3.0 and Turbo Basic 1.0 compilers. We will also discuss differences between these two compilers and the Microsoft "standard" for BASIC that most of us have used for years.

New Features for Turbo Basic

Turbo Basic uses an interactive editor/compiler to perform all tasks. Compiling source code requires that we enter the editor, load the source code file (this can be done from the command line also), and press Ctrl-R to compile and run the program. Compiled code can be saved to disk, but programs cannot be compiled directly from the DOS command line.

Turbo Basic's best new feature when compared to interpreted BASIC is the INTERRUPT function. This function allows the programmer to load an "array" of registers and call any interrupt service routine. All error handling must be done by the program, so this feature must be used with caution. Other features include:

CEIL	Force round up for integers
BIN$	Convert integer to binary string
OCT$	Convert to octal string
STATIC	Used in sub-prog only, no dim size required
UCASE$/LCASE$	Convert strings to upper or lower case
DO..EXIT DO	Available in QB 3.0 also
CASE	Available in QB 3.0 also
INLINE	Include assembler code as bytes into source
binary	File storage format,QB 3.0 feature also
8087	Supported

Turbo Basic does not permit precompiled libraries to be LINKed to a program. All BASIC sub-programs must be INCLUDEd into the source code and recompiled each time. The editor/compiler does allow you to specify separate directories for source code and included sub-programs. This feature facilitates including large lists of sub-programs and subroutines as long as they are all placed in this one "include directory." Chaining to program overlays is also available.

Turbo Basic gives the programmer access to assembler routines as either inline code or precompiled .COM files. The method for entering inline code is similar to interpreted BASIC's read-and-poke method for entering binary routines into source code. Turbo Basic has refined this process with the inline command. A typical format would look like this:

```
'main program segment
.
call  <procname> [(parameter list)]
.
SUB <procname> INLINE
$INLINE <byte by byte list of assembled code>
or
$INLINE "filename.COM"
END SUB
```

A utility program to convert assembler code to inline format is included on the program disk.

Turbo Basic handles all overhead when calling assembler routines as either inline code or precompiled .COM files. In QuickBASIC the number of variables passed to the subroutine must exactly match the number expected by the assembler routine; in Turbo Basic this number can vary (that is, you can have "optional" variables in a call).

Memory use is more flexible in Turbo Basic. Individual string arrays can be only 64K long, but each array uses a different 64-K block. In QuickBASIC we are limited to a total of 64K for all strings. Dynamic number arrays in both compilers are similar. Turbo Basic can access more than 64K with special programming using a segment descriptor.

Turbo Basic has a more restrictive "reserved word" test. The variable name LINE$, for example, is acceptable in QuickBASIC but fails as a reserved word in Turbo Basic.

New Features in QuickBASIC 3.0

QuickBASIC 3.0 includes an editor/compiler similar to Turbo Basic's, but also allows BASIC programs to be compiled from the DOS command line. External libraries can be accessed during DOS level compilation while a special library is needed for the interactive editor as we shall see below.

All previous code from QuickBASIC 2.0 is compatible with QuickBASIC 3.0. Several new programming features include:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WIDTH x,43	43 line text mode with EGA
CASE..SELECT CASE	
DO..LOOP	
CONST	
EXIT DO	
EXIT FOR	
DEBUG	Enhanced debug feature (very nice!)
8087	Supported (see below)
bugs	*Many small bugs fixed including CHAIN*
LINK	The latest Linker 3.06 is now supplied
binary	File storage format, TB 1.0 feature also

Microsoft was quick to add those features that were in Turbo Basic but not QuickBASIC 2.0. The command language is now almost identical with Turbo Basic's. The new CONST function allows the programmer to define a true constant with a name. In my programs I always use the variable names true% = –1: false% = 0 for readabilty. These *are* variables, however, and are accessed more slowly than constants.

With the command CONST true% = –1: CONST false% = 0, we can create true constant names that are global and produce more efficient code.

QuickBASIC 3.0 will compile programs outside the editor as did earlier versions and will accept .LIB libraries created and maintained with the Microsoft librarian. The librarian utility program does not come on the QuickBASIC program disks, which is a major shortcoming in my opinion since external compilation permits normal or /O (stand alone .EXE program) compilation with more than one library.

External .LIB libraries cannot be used with the QuickBASIC editor for in-memory compiling. In this case Microsoft's new QuickBASIC Compiler has a second type of object library called the USERLIB. This USERLIB is new to QuickBASIC Versions 2.0 and higher and works primarily with the interactive editor.

8087 Support

Turbo Basic was designed to work with the 8087/287 math coprocessor, but will operate without one (see "Execution Speed" below). The 8087 can be selected from the editor OPTIONS menu. Selecting "8087 required" will allow programs to execute slightly faster but will require that the chip be present for the compiled program to operate. The Turbo Basic editor will work either way. This is not true for the QuickBASIC editor as we will see in a minute.

The biggest enhancement in QuickBASIC 3.0 is the addition of 8087/287 support. New libraries are supplied for 8087 support as BRUN3087.LIB and BCOM3087.LIB that will use the 8087 if present. 8087 use can be overridden to use software

No, I'm not talking about which editor displays the most colors; I'm talking about the differences these two packages mean to your code.

How do we create user libraries? Microsoft supplies a program called BUILDLIB on the program disk. BUILDLIB will build a new USERLIB.EXE library from scratch. It cannot edit or maintain the library once it is created, however.

USERLIB libraries have several shortcomings the programmer should be aware of. First, to make any change to the library, no matter how small, you must reconstruct the *entire* library from scratch using BUILDLIB. This means that you must keep a copy of your object code files handy. Second, the USERLIB library that was used when compiling the program *must* be available when running the program. This means that there must be a copy of the USERLIB.EXE library file on the same directory as the compiled program. In addition, this USERLIB.EXE must be *exactly* the same as when we compiled. If *any* changes have been made to the library, our program may not run. This is more trouble than it's worth if you ask me!

The /O option cannot be used with USERLIB libraries or in memory compilation.

"emulation" with a NO87 environment variable for normally compiled programs (no /O).

For /O compiled programs using BCOM3087.LIB, 8087 "emulation" in software is possible if the program is linked to EMULATOR.OBJ. If the BCOM3087.LIB/O compiled program is not linked to EMULATOR, it will not operate without the 8087.

There are several more drawbacks to the 87 mode. The QuickBASIC editor/compiler is an entirely different program for the 87 version that will not run if a coprocessor is not installed. This means that you cannot develop 8087 compatible code unless you have one. Finally, the manual points out that there is a bug in DOS 3.2 that requires a PATCH program to properly run 8087 BASIC programs that you compile. If you're thinking of using your programs on more than one machine, of selling them commercially, you must make your users aware of this. The manual does not mention whether you can legally distribute this patch with your code.

Continued...

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Sound confusing? I would like your comments on how useful this arrangement is to your programming. Let me know.

Rounding and Integers

There is one outstanding difference between 8087 compatible compilers and the Microsoft "standard" that *every* programmer should be aware of. Turbo Basic uses IEEE format for 8087 compatibility as well as for storing numbers in files. The key words here are "8087 compatibility." To be fully compatible with the 8087, Turbo Basic handles rounding for integers by 8087 conventions. For the value ?.5, Turbo Basic rounds down on even whole numbers and rounds up on odd whole numbers. As you know, Microsoft BASIC (both interpreted and compiled) rounds up on .5 always.

Here are the differences:

Function	Results in			
	Microsoft (interpreted)	QBASIC (w/o 8087)	QBASIC (w/8087)	Turbo Basic (either)
INT(1.5)	1	1	1	1
CINT(1.5)	2	2	2	2
a%=1.5	2	2	2	2
a%=2.5	3	3	2	2
CEIL(2.5)	–	–	–	3
a%=0.5	1	1	0	0

As you can see from the a%= functions, Turbo Basic and QuickBASIC 8087 will give different results from the Microsoft standard for any function resulting in an even integer number, but only when the result is ?.5. Because of this subtle difference, no program can be converted to Turbo Basic or QuickBASIC 8087 from Microsoft interpreted or other compiled BASICs without close review of every integer equation! Borland was obviously aware of this difference and included the CEIL function to emulate Microsoft BASIC rounding.

Problem with Precision and Accuracy

Precision is the measure of how well a number reproduces itself. Accuracy is a measure of how well a number compares to the *true* value. Computers are well known for producing the same answer to an equation repeatedly, so they are considered to be very precise. Because of internal rounding, however, microcomputers do not always calculate the answer exactly correctly (say for the value of Pi to 99 decimal places). So we can say that microcomputers are not always accurate.

"Significant figures" refers to the number of digits that can be assumed to be correct in a number. The numbers 1.23 and 12.3 both have three significant figures even though the first has two decimals places and the second has one decimal place. In non-8087 Microsoft BASIC a single precision number is accurate to 6 decimal places and a double precision number is accurate to 15 decimal places. In 8087 mode this is raised by one to 7 and 16, respectively, because a different internal format is used by the processor for storing real numbers.

This means that even though the computer can print a single precision number as 1.23456789012344, this number is accurate to only 1.23456. The computer may reproduce this 15 digit number over and over again since it is very precise, but the trailing digits are meaningless. QuickBASIC is very careful to keep track of the number of significant figures for us. If a single precision number is multiplied by a double precision number, QuickBASIC clips the value and stores the result as a single precision variable since only the first 6 digits are accurate.

Turbo Basic was obviously written specifically for the 8087 math coprocessor. Turbo Basic code is optimized for full 16 digit precision in the IEEE format. This accounts for the magnificent speed of Turbo Basic for double precision calculations when running with an 8087 math coprocessor. This approach by the developers also accounts for several peculiarities in single precision calculations.

First, Turbo Basic appears to perform *all* calculations at full 16 digits precision. In BASIC terminology, the compiler handles all variables as double precision numbers. This can be tested by assigning a value to a double precision variable and then printing the variable's value:

```
'Turbo Basic double precision variable
A# = 1.0
B# = 1.23
Print A#, B#
 1        1.23
```

As you can see, trailing zeros are not printed even though the number is accurate to 16 or 17 digits. The PRINT USING statement would prove this.

Now let's assign these same values to a single precision variable in Turbo Basic:

```
'Turbo Basic single precision variable
A! = 1.0
B! = 1.23
Print A!, B!
 1.0      1.230000019073486
'(your number may differ slightly)
```

Here we find that Turbo Basic assigns the value to the variable B! accurately to 6 significant figures with a fractional error in

★ NOW AVAILABLE FOR C ★ QUICKBASIC ★
★ IBM BASIC/BASICA ★ TURBO PASCAL ★

The Aspen Systems Subroutine Editor (**ASE**) you can call from your programs. Design your own screen layouts - - update in several windows simultaneously. **ASE** is:

- ★ **TRANSPORTABLE** - can be configured for any keyboard and almost any computer or language under MSDOS.
- ★ **VERSATILE** - customize input for any application.
- ★ **COMPLETE** - with full screen edit capabilities and a wide variety of automatic conversions.
- ★ **EASY TO USE** - data and screen layouts described in a single map.
- ★ **FULLY DOCUMENTED** - including examples in BASIC, PASCAL, FORTRAN, C, and even COBOL.
- ★ **AFFORDABLE-**
  ASE is still only..... **$99**
  DEMO (See what ASE can do for you!)..... **$ 3**

The Aspen Systems Subroutine Package (**ASP**) - the companion package for **ASE** - with many functions unavailable or difficult to perform in high level languages. Like **ASE, ASP** is:

TRANSPORTABLE ★ FULLY DOCUMENTED
EFFICIENT ★ AFFORDABLE

ASP includes 150+ subroutines: conversions; sorts; string, bit, date/time functions; decimal arithmetic, and **MORE.**

ASP is still only..... **$130**

ASE and **ASP** include support and demo programs.

Prices are PPD (continental USA).
Colorado Residents add 3%.

P. O. Box 1163
Grand Junction, CO 81502
(303) 245-3262
VISA/MasterCard accepted

ASPEN SYSTEMS

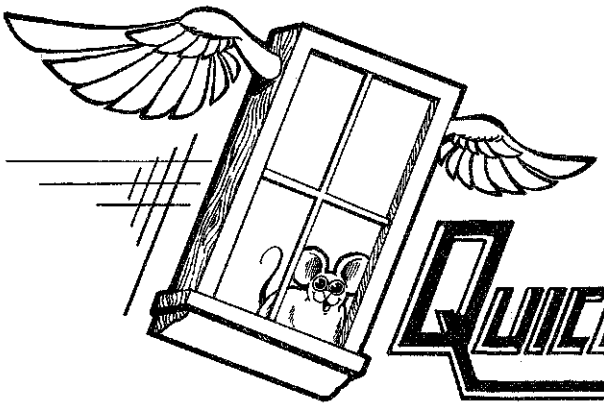
CP/M and MS-DOS are trademarks of Digital Research and Microsoft, respectively.

Number **86** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SAVE HOURS OF VALUABLE PROGRAMMING TIME!

Call for FREE Demo Disk



QuickWindows™

A low cost Window Management Library for BASIC

- Create windows, mouse/kb driven pop-up and pull-down menus, and powerful data entry routines quickly and easily.
- Fast! Written entirely in assembly language
- Efficient! Uses less than 24K of object code
- Full mouse interface for Microsoft compatible mouse.
- Device independence - Automatically detects video display.

Comes complete with a library of over 60 functions, a fully-documented manual, many programming examples, and a demo/tutorial program to show you how to use each of the functions. All for only $59. Add $20 for the QuickWindows™ assembly language source code.

No royalties are required for using QuickWindows™ in your programs and not copy protected. 30-day money back guarantee. For IBM/True compatibles and Microsoft QuickBasic or BASCOM, DOS 2.0 or later.

Call now for your
FREE demo disk!
401-274-5465
Visa, MC and Phone orders welcome.

Software Interphase INCORPORATED
5 Bradley Street, Suite 7
Providence, RI 02908-2304

Dealer discounts available.

Number **115** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal For IBM PC Programmers
PROGRAMMER'S Journal
DIRECT MAIL ORDER

ORIGINAL DESIGN
T-SHIRTS

BORN TO PROGRAM PJT101

NOW!
Soft Wear from Programmer's Journal!
Only $9.95 each, Postage paid. Classic PJ cartoons & cover graphics. Reproduced as dynamic Multi-color imprints on high quality 50/50 cotton T-Shirts. In sizes S, M, L, XL. Born to program? Send for yours today!

RS-232 HEAD
PJT103

PARALLEL PORTS
PJT102

COMMON C BUG
PJT104

BACK ISSUES

•Vol. 1, No. 1: Norton batch files and Pascal I/O; Program style manual; COBOL on the IBM PC; Benchmarks; System analysis; BASIC modules; Rounding error.

•Vol. 1, No. 2: Defining character sets; C86 BIOS calls; Troubleshooting; Copy protection; Pascal code; Input routines; Writing an assembler.

•Vol. 1, No. 4: Diskette directory via Pascal; MASM WHILEWEND; C/ASM interface; ASM stay resident routine; Structured programming; Software law; Pascal tutorial; Default values and variable names.

•Vol. 1, No. 5: Multi-tasking software review; Transporting C programs; Programmer's calculator; dBASE II bargraphs; Utilities; ROM BIOS equates; Pascal check program; Combining modules in Assembler; Structured programming in BASIC; Agreement outline.

•Vol. 2, No. 1: Freelance consulting; Sorting in C; Pascal text editor source code; CRT distress; Compiled BASIC; Assembly modules for C; Designing a relational database; Ultra Utilities; Punctuation and Style reviewed.

•Vol. 2, No. 2: Interview with Brian Kernighan; Understanding C through assembly language; Utility routines in C and assembly; Music in C; Automatic line number optimizing in BASIC; Assembly language tutorial; Full-screen input routines in BASIC; dBASE II printer control codes; Pascal tools reviewed; C programming support products.

•Vol. 2, No. 3: Software development environments; Creating a productive screen handling tool; Assembly tutorial; Generating graphics within windows; Composition and editing system for musical ideas; Hybridizing assembly and BASIC; Creating screens; Linking Assembler to Pascal and BASIC.

•Vol. 2, No. 4: Writing a custom ROM BIOS for PC DOS; RISC, Transputers, and mass storage; The future of disk protection; 8088 assembly macros; Random access files in BASIC; C utility routines you can buy; Using the PMATE macro language; Choosing a database manager; E/SP syntax-checking; Pascal editor reviewed.

•Vol. 2, No. 5: Developing the APX CORE Executive; From Assembler to C via a 211 byte .EXE main( ), Scroll control using BASIC and Assembler; Faster screen handling with BASIC and ASM; Random access files, Part 2; RTOOL, a PC-resident programmer's toolkit; dBkey reviewed; Pascal's pointer power.

•Vol. 2, No. 6: Periscope—building a hardware board for the IBM PC; Intel 80286 bugs by FNE; AT&T surprises all; Field Usage and pointer subtraction in C; Calendar functions for BASIC; Memory segmentation; Reduce your .EXE file sizes; Enhancing Debug; dBASE II to dBASE III? THINKTANK reviewed.

•Vol. 3, No. 1: November's Comdex; C—its uses and abuses; Using the C preprocessor; Memory segmentation examples; Index files, Part 1; Structured programming language; Periscope, a symbolic debugger.

•Vol. 3, No. 2: UNIX! (again); Artificial intelligence; Doctor C's tips—pointers to functions; Calendar date routines in BASIC; Calculate record layouts, automatically!

•Vol. 3, No. 3: NAPLPS and the professional programmer; Instant-C; Printer control from a batch file; Repositioning the stars in Starfinder; Reducing EXE file size through data positioning; tools for better programming.

•Vol 3, No. 4: More UNIX stuff; C programming books, Part 1; C code excerpts, Part 1; How to cure the "dreaded locked keyboard" condition; Calling MS DOS from Turbo Pascal; DOS 2.x sort and find filters; the SET Tools.

•Vol 3, No. 5: Windows done here; C programming books, Part 2; C code excerpts, Part 2; A menu-driven PC using DISPLAY.COM; Turbo Pascal and MASM: making them work together; dBASE II routines, Turbo Pascal 3.0; TopView Programmer's Toolkit.

•Vol. 3, No. 6: About C's HLL overhead; How to recover all your costs; Report on the Comdex and Usenix shows; The Santa Cruz Operation XENIX for the IBM PC; Common C bugs; CTOOLS2; Turbo Pascal 3.0 and subdirectories; More dBASE II routines; Clipper reviewed; A better way to interface BASIC and Assembler.

•Vol. 4, No. 1: FNE on PC/AT hard disk problems and plunging 64K DRAM prices; AUTOINC explained; Sorting using an AVL tree; Combining binary trees; Using DOS to call programs from Turbo Pascal; An operating system for the people; Final installment of Brett Salter's dTOOLS; Two microcomputer graphics books reviewed.

•Vol. 4, No. 3: FNE—another RISC machine, the paging game, software viewpoint, programmer talent, and Aloha, 32032; Sun Microsystems' next generation PC, Consulting—reusable modules; INT24.PAS, a critical error handler for Turbo Pascal; Memory model independence in Assembler; An approach to windowing for the assembly language programmer; Proposed ANSI C Standard, Part 2; Microsoft and Lattice C 3.0 reviewed; Accelerator boards for software development reviewed.

•Vol. 4, No. 4: Hal W. Hardenbergh on adult toys, tomorrow's micros, HALGOL update, benchmarks, and RISCs; Frank D. Greco on parallel computers, RISC, UNIX; Consulting—efficient custom programming; DOS 2 file handles and file I/O; Programming the WordStar document file conversion utility; More optimizing for speed; Teach an old compiler a new trick; Memory resident programs in C; Tips on formatting with BASIC PRINT USING statements; The structure of dBASE II and III file headers.

•Vol. 4, No. 5: Hal W. Hardenbergh on 32-bit PCs, the 68040, Weitek's data-path part, etc.; Frank D. Greco on large screen monitors, language translators, UNIX, and more; Prolog intro, Turbo Prolog reviewed; Consulting—tax traps, consulting styles, nest eggs; ASM—Search and Display; ASM—the GROUP statement; The proposed ANSI C Standard, Part 3; C—measuring stack usage and Lattice C's startup code; Short boards reviewed.

•Vol. 4, No. 6: DTACK GROUNDED; Unix Books, Part 1; How many amoebas at one o'clock; DOS functions 45H and 46H; DOS Terminate and Stay Resident; Turbo Pascal inline code; Pascal—Assembler souce code for DOS function calls; Alignment problems in C and managing memory allocation; Proposed ANSI C Language Standard, Part 4; The Canon A1 laser printer; Genifer dBASE applications generator reviewed; Advanced Trace86 debugger reviewed.

•Vol. 5, No. 1: DTACK GROUNDED; The new DOS; Marketing and rules to live by; File redirection; Inside the EGA; String allocation in Pascal; ANSI C Language Standard, Part 5; More tips on formatting BASIC data.

•Vol. 5, No. 2: DTACK GROUNDED; Starting your own software house; Pink noise; TSR graphics print screen utility; Coding guidelines for resident programs; Parallel processing with the EGA; C portability in practice and using header files; Proposed ANSI C Language Standard, Part 6; Indexing in BASIC; A first look at Turbo C.

•Vol. 5, No. 3: Starting your own software house—publicity; Unix books, part 2; Consulting—merger mania; EGA data control; Modula-2 for Pascal lovers; Optimizing Turbo Pascal; ANSI C language standard; Coping with ANSI C and Forcing File Updates, Formatting BASIC output with buffered PRINT USING.

•Vol. 5, No.4: Starting your own software house—technical support; Unix books, part 3; Consultant productivity and effectiveness; EGA set/reset capabilities; ANSI C language standard; Character tables and string functions in C; MWWIND—a menu with windows utility; Variable passing and hiding in QuickBASIC 2 subprograms.

To order, just use the convenient form located on the perf insert, at right. If the card is missing, you may place credit card orde by calling our office at:
(503) 747-0800

PRODUCED AND BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the remaining digits due to IEEE conversion error. Surprisingly, Turbo Basic prints the full 16 digit value of the variable even though the variable is accurate to only 6 significant digits (5 decimal places in this case).

This could be a bug in the internal printing routine of the compiler. Surely the routine should print only 6 significant digits and round or truncate the last digit. This would produce the number 1.23000, which is correct.

To test the print routine we can run another little program. If we assign 1.23 to both a single precision and a double precision variable, these should be equal even if the internal print routine is not functioning correctly.

```
'Turbo Basic accuracy
A! = 1.23
A# = 1.23
if A# = A! then
      print "Same Value"
   else
      print "Different Values"
enf if
```

Different Values

That's right, in Turbo Basic 1.23 ≠1.23 if the variables are different precision! Assigning 1.23 to A! and B!, on the other hand, *does* return equal values because the compiler produces the same error on each conversion. The compiler is very precise but not very accurate.

If you're wondering, QuickBASIC clips the precision of mixed precision equations to the least precise variable.

```
'QuickBASIC accuracy test with 8087
A! = 1.23
A# = 1.23000001
if A# = A! then
      print "Same Value"
   else
      print "Different Values"
enf if
print A!, A#

Same Value
 1.2300000190734860   1.23000001
```

The precision of the equation A! = A# must be reduced to the lower precision of the two variables for the comparison since QuickBASIC reports that the two variables are equal even though the double precision variable A# is not equal to A! beyond the 6 digit precision of A!. This is an accurate result within the precision of the numbers we are comparing.

Two More Annoying Features

Turbo Basic reports large double precision numbers differently than the Microsoft standard. A large number such as 1.23×10^{23} is normally reported in Microsoft exponential format as 1.23D+23 where the "D" designates the exponent of a double precision number, and "E" a single precision number. Turbo Basic reports this double precision number as 1.23E+023, using the "E" designator for both single and double precision values.

I discovered this difference in a routine I wrote to round large numbers. The program was checking for the "D" in the string version of double precision numbers. Also notice that the exponent is reported as 3 digits to support the wider dynamic range of IEEE numbers. QuickBASIC on the other hand reports only the number of exponent digits needed.

Another feature of Turbo Basic is even less obvious. In Microsoft BASIC, setting the printer line width to 255 with the WIDTH statement will inhibit line feeds for graphics plots. This doesn't work in Turbo Basic. For some reason the width must be much larger. I found that WIDTH 1024 will work. A call to Borland Support confirmed my finding.

Execution Speed

Actual execution speed of individual commands does not tell the real story of how fast your particular program will operate. Turbo Basic optimizes compiled code better than QuickBASIC, but was written specifically for the 8087 math coprocessor. Turbo Basic is marginally faster in the 8087 mode than QuickBASIC for a large scientific program that I recently tested. If you're not using the 8087 mode, however, QuickBASIC will in general execute measurably faster than Turbo Basic.

Both compilers do produce faster executing code than any of the "new" generation of BASIC compilers including Better Basic by Summit Software, True BASIC by Addison-Wesley, and ZBasic by Zedcor. Each of these compilers excels in the specific applications areas for which it was designed, but in general does not outperform either Turbo Basic or QuickBASIC.

It is even more difficult to compare QuickBASIC and Turbo Basic against C coded programs. The general consensus is that QuickBASIC is as fast as or faster than C in all areas except string handling. Many of the commands written into BASIC in machine language must be written as C source code libraries and compiled with the C program. It's not surprising that many of these standard BASIC features are slower in C.

Comments

Turbo Basic is not as "Microsoft compatible" as it appears from first sight. All converted software will need to be checked for real to integer conversions, equations containing variables of differing precisions, and printer width control routines. Programs running under QuickBASIC 8087 will need to be checked only for real to integer conversions.

If you uncover true bugs in either QuickBASIC or Turbo Basic, be sure to report them directly to the manufacturer. If you discover other peculiarities, as I have, forward them to me. If I receive a sufficient number of these interesting "features," I will present a follow-up article in *PJ*. □

Paul P. Dymerski has worked with BASIC for over ten years and has beta tested both QuickBASIC and Turbo Basic. As developer of the Finally! line of BASIC utilities, marketed by Komputerwerk, Inc., he has had opportunity to put many of the BASIC compilers through their paces. His company, gamut, does custom program development and specializes in scientific laboratory "computerization" (computer integration of the entire laboratory process from data acquisition to report generation). Readers who want to contact Mr. Dymerski may write to

Paul P. Dymerski
gamut
9616 Northbridge Court
Richmond, VA 23235

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BASIC Programming Tools

There's nothing basic about these professional programming utilities.

Whether you're a seasoned expert or just starting out, we can help you create programs that run faster, work harder, and simply look better. We have built our reputation on customer satisfaction by providing expert advice and quality technical support. All Crescent Software products include source code, demonstration programs, clear documentation, and a 30-day satisfaction guarantee.

- **QBase** is a superb screen designer *and* full-featured relational database. Because we include complete BASIC source code, QBase can be customized to suit your needs. Besides its database capabilities, QBase is outstanding for creating custom titles, help screens, and product demos, and includes a versatile slide-show program. **$99**

NEW!

- **QBase Report** enhances QBase by generating reports with multiple levels of sorting, automatic record updating, browsing, and transaction posting. As a special bonus, QBase Report includes a complete, ready to run, time billing application for computer professionals. **$69**

- **QuickPak** contains more than 65 essential routines for BASIC programmers. Included are programs for windowing, access to DOS and BIOS services, searching and sorting string arrays, creating pull-down and Lotus™ menus, accepting data input, and much more. QuickPak is loaded with examples and tutorial information, and comes with a clever tips and tricks book, plus *The Assembly Tutor* — *a* complete guide to learning assembly language from a BASIC perspective. **$69**

- **By Customer Demand — QuickPak II** More than 30 additional tools, including disk and printer tests to eliminate the need for On Error in your programs,

and a multi-line text input routine that lets you put a note pad with full word wrap anywhere on the screen from within your programs. Other routines include binary file access, more menus, multiple screen save and restore, continuous time display, automatic box drawing, and much more. (Available for use with QuickBASIC *only*.) **$49**

NEW!

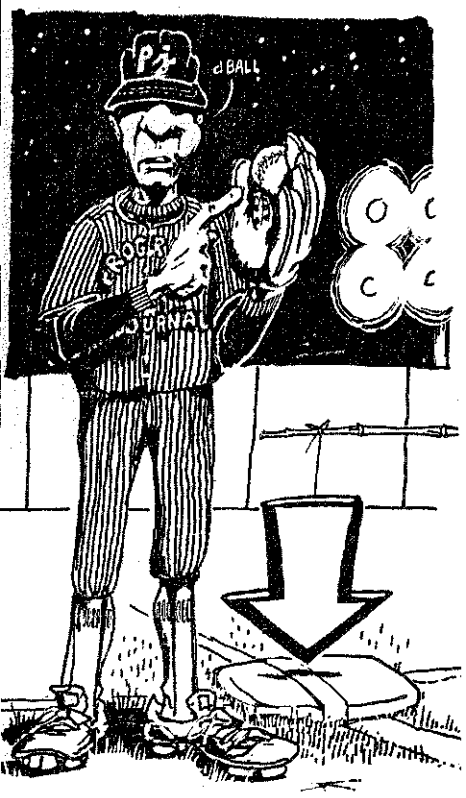
- **GraphPak** is an extensive collection of routines for displaying line, bar, and pie charts automatically within your programs. It will create 3-D charts with manual or automatic scaling, titles and legends in any size or style, as well as scrolling and windowing *in graphics*. GraphPak also comes with a sophisticated font editor for customizing your own character sets. **$69**

CRESCENT SOFTWARE

64 Fort Point Street, East Norwalk, CT 06855
(203) 846-2500

Separate versions are available for Microsoft QuickBASIC and Borland Turbo Basic — *please* specify when ordering. No royalties, not copy protected, of course. We accept Visa, M/C, C.O.D. and Checks. Add $3 shipping and handling, $10 overnight and foreign, $25 2nd day foreign.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



dBASE
(see arrow)

dBASE Programmers

are you ready?

dB2→c

"C" IS READY FOR YOU!

dB2c Offers:

- Complete implementation of dBASE III+ functions and commands in C source code.
- Interfaces for Lattice's dBC and Faircom's c-tree.
- Complete source code
- Support for the following C Compilers: Microsoft, Lattice, Aztec, DeSmet, and Instant-C.
- The freedom and peace of mind that only source code can offer.
- Works on color, monochrome, and EGA monitors.

dB2c Toolkit $299.

Call or Write:

DAVID J. MARSH

SOFTWARE CONNECTION, INC.
POB 712, Ely, MN 55731
(218) 365-5097

Number **93** on the Reader Service Card

TRUE MULTITASKING

With

MultiDos Plus

"multitasking for the IBM-PC."

Ideal for developing applications in process control, data acquisition, communications, and other areas. Check these features which make **MultiDos Plus** an unbeatable value.

- Run up to 32 programs concurrently.
- Your software continues to run under DOS. No need to learn a new operating system.
- Use the compilers you already have. Supports software written in most languages.
- Operator commands to load/run programs, change priority, check program status, abort/suspend/resume programs.
- Programmatic interface via INT 15H for the following.
  - * Intertask message communication. Send/receive/check message present on 64 message queues.
  - * Task control by means of semaphores. Get/release/check semaphores.
  - * Change priority-256 priority levels.
  - * Suspend task for specified interval.
  - * Spawn and terminate external and internal tasks.
  - * Disable/enable multitasking.
  - * and more!
- Independent foreground/background displays.
- Access to DOS while applications are running.

Hardware/Software Requirements

IBM PC/XT/AT or true clone. Enough memory to hold **MultiDos Plus** (48 KB) and all your application programs. Also may need 4 or 16 KB memory for "hidden screens" for each active task. MS-DOS (or PC-DOS) 2.0 or later operating system.

only: $24.95 *OR* $99.95 with source code

Outside USA add $5.00 shipping and handling.

Visa and Mastercard orders only call toll-free: 1-800-872-4566, ext. 350., or send check or money order to:

NANOSOFT
13 Westfield Rd, Natick, MA 01760
MA orders add 5% sales tax.

Number **109** on the Reader Service Card

HELP

is at hand

HELP/Control™ — an on-line help system for the IBM-PC. **HELP/Control** includes **HELP/Runtime, HELP/Popup** and our help screen compiler.

With **HELP/Runtime**, a few simple subroutine calls add context sensitive on-line help to your application. **HELP/Runtime** includes tested interfaces for C (Microsoft and Lattice), Pascal (Microsoft and Turbo), IBM BASIC (Interpreter and Compiler), Microsoft FORTRAN, COBOL (IBM and Realia) and assembler.

Use our concise screen definition language to build your help files. You define the bold captions on your help screens and specify the links to other screens. If you have existing documentation files, we supply a program which automatically marks them up to get on-line quickly. You can put an entire user or reference manual on-line, completely accessible to the user at all times.

HELP/Control also includes **HELP/Popup**, which provides memory resident access to your custom help screens. Use it to document dBase and 123 applications. **HELP/Popup** uses the same help files as **HELP/Runtime** and operates the same from the user's point of view, allowing you to provide a consistent on-line environment across diverse applications.

The complete package (software, on-line manual, printed manual, and demo programs) costs $125.00 and includes a royalty-free license to add **HELP/Runtime** to your applications and a license to make 25 copies of **HELP/Popup**. A demonstration diskette, including the on-line manual, costs $15.00. To order, or for more information (including dealer, multiple-copy and site-license pricing) call MDS at 207/772-5436. We accept MasterCard and VISA.

New options with release 1.20:

- **Use Dan Bricklin's Demo Program to build your help system. Our translator turns it into a HELP/Control source file.**
- **Include graphics in your help screens.**
- **Mouse support.**

MDS

MDS, INC., P.O. BOX 1237, PORTLAND, MAINE 04104

Number **69** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Graphic Standards

X Windows
MIT Microcomputer Center
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
(617) 253-5851

Wouldn't it be nice to have a "real" graphic standard to program for? Are you tired of struggling to become fluent with the latest EGA deluxe super plus enhanced graphic board introduced by Megapixels Interstellar Conglomerate? Well, help may be on the way.

This spring, a number of computer manufacturers announced a commitment to and support for the X Window system, a public windowing graphic standard for the computer environment. Major companies agreeing to the standard included Adobe Systems, Apollo Computer, Applix, Dana, Data General, DEC, Hewlett-Packard, MassComp, Siemens, and Sony and Stellar. SUN a few weeks later also agreed to incorporate the standard on their own network workstations. The X Window standard has been under design and development by Project Athena Group and the Laboratory for Computer Science at MIT since 1984. Project Athena is exploring the use of high performance workstations in the educational environment. The windowing system is designed to display multiple applications on a screen at the same time and has been extended to support network environments. MIT and the computer vendors agreed to promote X Window as an industry standard for the display of graphic information in networked environments. The companies also backed a cooperative effort to extend the X Window specification to include a set of high-level programming support tools.

From the beginning X Window was developed with two primary goals—make it transparent across a multi-vendor network and transportable for applications software. It was specifically designed to support diverse hardware and provide machine independent interfaces. X Window is feature-rich and provides a comprehensive set of graphic capabilities for manipulating screens, fonts, and objects and interfacing with various input devices. The protocol is also extensible. Given the level of industry support, MIT might just pull it off.

X Window is a public resource, nonproprietary standard. MIT distributes complete X Window source code for the cost of the medium, without any license restrictions. MIT is currently distributing Version 10, Release 4 of the X Windows protocol specification and support tools. By mid-September, Version 11 will be available. A two page flyer is available for free. A copy of the documentation manual describing the standard is available for $9. The full X Window source code (for a variety of environments) along with two printed manuals (standard and protocol manuals) is available for $100. The sources are provided on a 9-track magnetic tape (1600 bpi) in the Unix Tar format (don't get stuck now).

New Operating Systems

Microsoft OS/2 Software Development Kit
Microsoft Corporation
Dept. MS OS/2 SDK
16011 NE 36th Way
Redmond, WA 98073
(206) 882-8654

Since April, Microsoft and IBM have been vigorously promoting OS/2 (Operating System/2), the new operating system for 80286/80386-based personal computers. MS OS/2 is Microsoft's next generation operating system for the IBM PC AT and compatibles. Microsoft has already held a few seminars for developers, which cost $3000 per person (plus sales tax) and include a copy of the OS/2 Software Development Kit (SDK) when it becomes available. Microsoft is supposed to begin shipping the OS/2 Developer's Kit to developers in August. The OS/2 SDK is a "comprehensive set of prefinal release (beta test) versions" of Microsoft's new 80286 and 80386 system software. Don't expect a rock solid product, eh!

The Software Development Kit includes the following:

Operating System/2 (including Windows presentation manager)
Operating System/2 LAN Manager
Microsoft C Compiler for OS/2
Microsoft Macro Assembler for OS/2
Microsoft DIAL (electronic mail support)

The DIAL option (not the deodorant soap), Microsoft's technical support service via electronic mail, may be enhanced with direct phone support for an extra $1000/year. Regular updates to the kit are included in the licensing fee for the SDK.

The Developer's Kit also includes attendance at a Developer's Conference. Aside from the previous seminars, additional Developer's Conferences are scheduled on September 22–24 (Anaheim, CA) and October 21–23 (Dallas, TX). More will likely follow.

Unix V/386
AT&T Unix Source Licensing
Data Systems Division
P.O. Box 2500
Greensboro, NC 27420
(800) 828-UNIX

Sandra Duncan
Intel Corp.
3065 Bowers Avenue
Santa Clara, CA 95051
(408) 496-4580

The first new operating system for the 80386 arrives. AT&T and Intel announced the availability of Unix System V source code licenses for the 80386. System V/386 release 3.0 was developed by Intel as a contractor to AT&T to port Unix to the 80386. Intel was obviously interested in speeding up and facilitating this port of Unix to their latest microprocessor. Intel subcontracted with Interactive Systems (Santa Monica, CA) for assistance with the port. AT&T apparently has the port back from Intel and is readying distribution and sales. The initial release date of July 7 was pushed to later in July. The AT&T licensing program manager had been on the job only three days and was not exactly on top of the details.

At press time, AT&T had still not determined the pricing schedules although earlier press releases had noted $20,000 for the source code license. A "certified" version of the software should be available in August. A source code licensee would also likely acquire rights to sub-license Unix V/386, which limits them to distributing object code or binary sublicenses. The source code license includes the entire Unix system, not a subset or just the kernel. The source is normally distributed on 9-track magnetic tape (1600 bpi) in the Unix CPIO format. Several companies have already secured source licenses and will likely soon announce the "real" availability of binary Unix V for the 80386. These companies include the following: Interactive Systems (213/453-8649), Bell Technologies (415/659-9097), and Microport(408/438-UNIX). In addition, OEMs such as Prime also have source code licenses for their own machines.

IBM PS/2 Manuals

IBM Technical Directory
P.O. Box 2009
Racine, WI 53404
(800) 426-7282

At last, the manuals that developers and "hackers" have been waiting for. IBM has begun releasing the much anticipated BIOS and technical reference manuals for the new

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VEDIT PLUS

#1 PROGRAMMABLE EDITOR

Call 1-800-45-VEDIT for FREE Fully Functional Demo Disk

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS easily satisfies the most demanding computer professional.

Try a Dazzling Demo Yourself.

The free demo disk is fully functional—you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial—you experiment in one window while another gives instructions.

The powerful "macro" programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a "macro"—it shows that no other editor's "macro" language even comes close. And VEDIT PLUS is only 40K in size.

Go ahead. Call for your free demo today, You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Only VEDIT PLUS is this Flexible.

The installation lets you pick from closely emulating the keyboard layout of Word Perfect, WordStar and others. Or you can easily create your own layout and even your own editing functions. Supports any screen size—you pick screen colors and attributes.

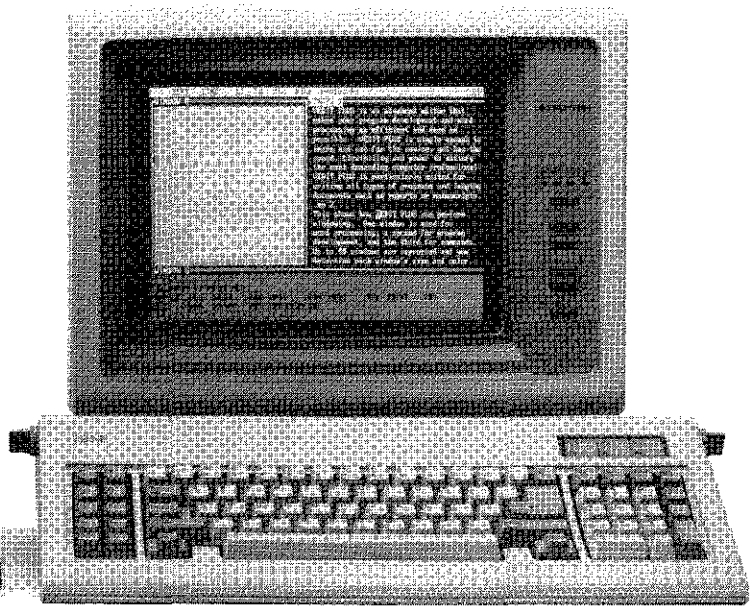
Supports the IBM PC, XT, AT and PS/2. Also supports MultiLink, PC-MOS/386, Concurrent DOS and most networks. Also available for MS-DOS, FlexOS (protected mode), CP/M-86 and CP/M. (Yes, we support windows on most CRT terminals, including CRTs connected to an IBM PC.) Order direct or from your dealer. $185.

Special: VEDIT (single file, no windows) for CP/M—$49.

- **Fully Network Compatible**
- **Call for XENIX-286 version**
- **30 Day Money-back guarantee**

Compare Features and Speed

	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	Unlimited
Multiple file editing	20+	2	No	20+
Windows	20+	2	No	20+
Macro execution window	No	No	No	Yes
Pop-up menus	No	No	No	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Automatic processing of Compiler errors	Yes	No	No	Yes
"Cut and paste" buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
Convert to/from WordStar	No	No	No	Yes
On-line calculator	No	No	No	Yes
Configurable Keyboard	Hard	No	Hard	Easy
43 line EGA support	Yes	No	No	Yes
Manual size/index	250/No	42/no	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min	34 sec	1:07 min	6 sec
Pattern matching search	20 sec	Cannot	Cannot	2 sec
Pattern matching replace	2:40 min	Cannot	Cannot	11 sec



VEDIT and CompuView are registered trademarks of CompuView Products, Inc. BRIEF is a trademark of UnderWare, Inc. PMATE is a trademark of Phoenix Technologies Ltd. Norton Editor is a trademark of Peter Norton Computing Inc. MultiLink and PC-MOS/386 are trademarks of the The Software Link, Inc. CP/M and FlexOS are trademarks of Digital Research. MS-DOS is a trademark of Microsoft.

*Also available for TI Professional, Tandy 2000, DEC Rainbow, Wyse WY700 and others.
*Demo disk is fully functional, but does not readily write large files.

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299, TELEX 701821

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PS/2 series computers. The manuals can be ordered through the mail or by phone using major credit cards. The orders are placed with a firm in Wisconsin (ordering service hired by IBM), and the manuals are shipped from their warehouse in Fayetteville, North Carolina. UPS Blue Label shipping may be requested at no extra charge. Shipping and handling is $5 per order, along with state sales tax. A list of these and other IBM PC manuals may be requested at no charge. Be forewarned that the PS/2 *Technical Reference* manuals do *not* contain schematics. They do contain pinouts for connectors, chip descriptions, input/output ports, and programming interfaces. These new manuals are much more expensive than the original PC or AT technical manuals. The new BIOS manual, for example, still comes in the 2″ thick slip binder but is only ½″ thick (200 pages) and contains no BIOS listings. The following manuals were available in mid-July, but check with the ordering service for an up-to-date list of new publications.

BIOS and Technical Reference Manuals
PS/2 Bios Interface Manual (Part #68X2260) $75
PS/2 Model 30 Technical Reference (Part #68X2201) $75
PS/2 Model 50/60 Technical Reference (Part #68X2224) $125
PS/2 Model 80 Technical Reference (Part #68X2201) $125

An additional 11 hardware, maintenance, and service manuals for the PS/2 line are also available.

TASKVIEW
ROCK SOLID MULTITASKING!

Packed with the power you need, TASKVIEW takes you beyond the limits of DOS.

- Communicate while you edit
- Compile while you print
- Load up to 5 megabytes of programs
- Manage resident utilities
- Time-slice multiple jobs
- Cut & paste between programs

TASKVIEW lets you load up to 10 of your favorite applications and switch between them at a keystroke. They can even continue to run while you work on something else! TASKVIEW is the BEST multitasker you can buy. Just ask our customers.

"Thanks for a wonderful product!"

"TASKVIEW is the only multitasker I own which runs efficiently & correctly both on my Zenith Z-151. . . and on my Orchid 286e . . . I also have DoubleDOS, DESQview, TopView, Windows, and Concurrent PC-DOS".

"Fantastic! What DoubleDOS, MS-Windows, and all the other 'stuff' should have been!"

"My BBS is now up 24 hours, 7 days a week thanks to TASKVIEW."

"Thanks! I needed that!"

TASKVIEW requires an IBM PC, XT, AT or Jr compatible, and PC or MS DOS 2.x to 3.x. To get your copy, call toll free:
(800) 367-0651
or send $79.95 + $3.00 S&H ($8.00 Intl.) to:

Sunny Hill Software
PO Box 55278, Seattle, WA 98155-5278
For more information call (206) 367-0650
DoubleDOS trademark Softlogic Solutions, Concurrent PC-DOS reg. trademark Digital Research Corp., MS Windows trademark Microsoft Corp., DESQview trademark Quarterdeck Office Systems, Inc. TopView trademark IBM Corp.

Number **85** on the Reader Service Card

Datalife 3.5" Microdisks
Verbatim Corporation
1200 W. T. Harris Blvd
Charlotte, NC 28213
(704) 547-6500

Verbatim Corporation, in rapid response to IBM's Personal System/2 introduction, has unveiled a 3.5" floppy disk compatible with the new generation of computers with twice the capacity of standard microdisks. The Verbatim Datalife 2-megabyte microdisk is a double-sided, 135 tpi product (80 tracks per side) compatible with the IBM PS/2 models 50, 60, and 80. Formatted disk capacity is 1.44 megabytes. Unfortunately, this new 3.5" technology does not come cheap. Have you been buying 5.25″ generic disks for 30 to 50 cents each, name brand disks for maybe twice that? These Verbatim 2-megabyte microdisks will be packaged in boxes of ten with a suggested retail price of $125 per box. Might want to wait before converting your 300 volume 5.25″ disk library to microdisks.

IBM PS/2 Sales
A number of items in the trade press have focused on the discrepancy between IBM's stated figures for PS/2 shipments and polls of dealer sales. IBM has stated that 250,000 units have been "shipped" in the first two months since April while queries to dealers identify maybe 40,000 to 50,000 sold. Either IBM has sold an awful lot of PS/2s to corporate accounts or someone is fibbing. The Model 80 (80386) was to ship a month earlier than estimated in June; however, dealers I have spoken with are still waiting for their floor models in July. I recently priced both the IBM Model 80 and the Compaq 386 with my local dealer. With discounts, the prices were surprisingly close for the 40 megabyte units, but favored Compaq for the larger hard disk models. By the way, the 80387 math chip is priced $100 cheaper from IBM than from Compaq, if you need one. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "Recommended. If you need a general purpose PC programming editor, look no further." His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

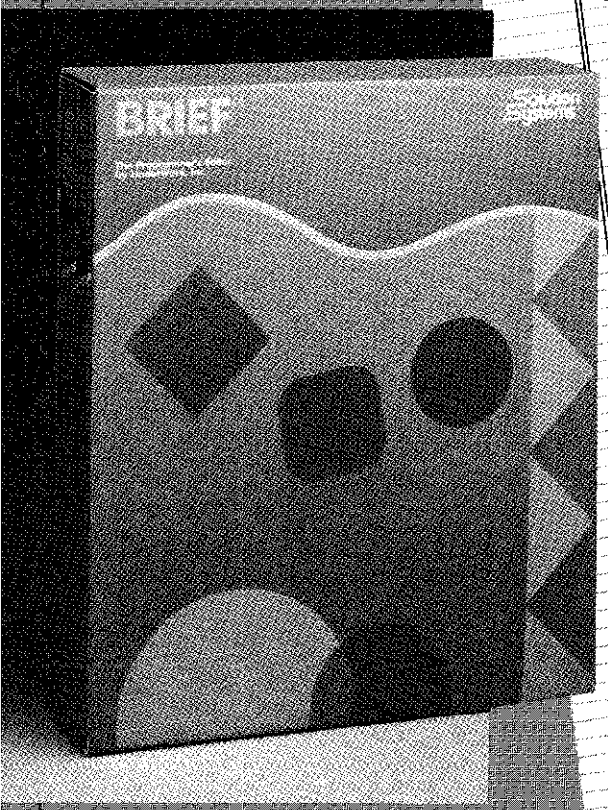
One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.

TO ORDER CALL: 1-800-821-2492 (in MA call 617-337-6963)

As always, BRIEF comes with a 30-day money-back satisfaction guarantee.

Solution Systems™

541 Main Street
Suite 410
So. Weymouth, MA 02190
(617) 337-6963



Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions and the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF SO popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.
BRIEF is a trademark of Underware, Inc.
Solution Systems is a trademark of Solution Systems.

Number 47 on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

CODE IN THIS ISSUE —

The code listings appearing in this issue of ***Programmer's Journal*** are available on:

Turbo Source Search BBS
P. O. Box 876
Scituate, MA 02066
617-545-9131 *RS# 35*

See ad on page 15

DOS USER'S GROUP

- Quarterly Newsletter
- DOS Tutorials
- Third Party Reviews
- OS/2
- MS Windows

Annual Membership $25
Canada & Europe $35

DOS User's Group
Box 26601
Las Vegas, NV 89126 *RS #128*

COMMUNICATIONS

Custom solutions for Asynchronous, Bisynch, SDLC, Multiline, Multitasking. Unattended Remote Polling Package with Scheduler, Mailbox, Remote Batch,and Full Duplex Protocol. Many communications utilities.

ModemArt
P. O. Box 0081
Richardson, TX 75083-0081
214-530-9572 *RS #130*

GreatBac — $49.95
FAST HARD DISK BACKUP
for PC/XT/AT & Compatibles

Fast, easy to use, affordable. File oriented backup, append, compare, restore XT Backup 10 Mbytes in less than 9 minutes: AT, 20 Mbytes in 9 minutes. Not copy protected. User modifiable defaults. Wildcard file names. MC/Visa/Check/MO Accepted.

Sellers & Company
3275 Independence Pkwy, Ste 153
Plano, TX 75075
214-964-5067 *RS # 129*

INSTANT ACCESS

To Any Directory without typing full path names. The COMMAND System automatically generates Short Names for each directory program. Fast Programs; functions DOS forgot. Super Directory program. Fantabulous SEARCH Program. 30-day Money-Back Guarantee. **Buy Now: Introductory Price of $75.** Send Check or Call with VISA oc MC order. Free U.S. shipping.

Compumagic, Inc.
P. O. Box 437
Severn, MD 21144
301-969-8068 *RS # 133*

POCKET REFFERENCES FOR C PROGRAMMERS, UNIX AND DOS USERS

C Library Reference — $6.00
MS-DOS Reference — $3.00
UNIX Command Summary —$6.00
Exploring UNIX Environment — $16.95
(Specify System V, BSD 4.2/4.3, or XENIX V)

Other references, UNIX software, and consulting available.

To order or send for our free catalog write or call:

SSC, Inc.
P. O. Box 55549-P
Seattle, WA 98155
206-367-UNIX *RS #131*

THE POINTER

The Memory Resident Help System and Demo Program Generator

Custom fit this flashy, menu driven help system to any new or old program without without programming, instantly! With the Demo Program Generator, create LIVE DEMOS for any program that runs a picture show. Actually run your applications LIVE through the demo windows with The Pointer. Rush $129.95 U.S. to:

Stau Enterprises, Inc.
3650 Kaneff Cres. Suite 1808
Miss. Ont. Can. L5A 4A1
416-270-6673 *RS # 134*

CONVERT BASIC TO C or PASCAL

BAS_C™ converts BASICA to C which car directly compiled by any C compiler. BAS_PAS™ converts BASICA to Turbo Pascal™. Both BAS_C™ and Bas_PAS™ transform the spaghetti code to the scoped indented, structured program automatically. Deadcode. Graphic, Random file.

Sample Conversion: Send your own BASI program (1 line / 1¢) and receive the printo

Demo disk $5.00

Gotoless Conversion
P.O. Box 835910
Richardson, TX 75083
214-404-1404 *RS #127*

TURBO PASCAL PROGRAMMERS

The SOURCE CODE for TURBO Pascal is available! For a remarkably low price users now have fully commented source code for TURBO Pascal v3.01A (binary arithmetic). This includes the run-time library, editor, an compiler. Everything you need is included, an asembler! Free Info.

Price: IBM-PC $67.50, Z-80 $45.00 (specify disk reqd.)

C.C. Software
1907 Alvarado Ave. #B
Walnut Creek, CA 94596
415-939-8153 *RS # 132*

ADVERTISER INDEX 5.5

Reader Service#	Advertiser Name	Page Number
86	Aspen Systems	54
30	Austin Code Works	47
51	Az-Tech Software, Inc.	47
2	Blaise Computing	3
3	Borland International	C4
38	C Ware	18
132	CC Software	64
56	CompuView Products, Inc.	61
133	Compumagic, Inc.	64
*	Creative Programming Consultants	46
83	Crescent Software	58
124	Custom Real-Time Software	22
128	DOS Users Group	64
70	E.C.I.	32
8	Ecosoft, Inc.	1
*	Gimpel Software	27
127	Gotoless Conversion	64
57	Hawkeye Grafix	32
58	Interface Group, The	23
63	Laboratory Microsystems, Inc.	34
80	Mace Utilities	7
92	Manx Software Systems	9
69	MDS, Inc.	59
45	MetaWare	47
13	Micromotion	26
104	MicroWay	13
121	Misosys, Inc.	40
94	MMC AD Systems	47
130	ModemArt	64
109	NanoSoft	59
16	Opt-Tech Data Processing	41
102	Oregon Software	31
6	Periscope Company, The	50
*	Programmer's Journal	56
82	PS Business Systems	41
*	Quaid Software Limited	43
46	Quilt Computing	33
*	Raima Corporation	51
129	Sellers and Company	64
93	Software Connection	59
98	Software Family, The	5
108	Software Garden, Inc.	19
115	Software Interphase, Inc.	55
110	Softway, Inc.	33
47	Solution Systems	63
131	SSC	64
134	Stau Enterprises	64
85	Sunny Hill Software	62
74	Sydetech	C3
126	Tangent Designs, Inc.	36
117	Tom Rettig Associates	8
35	Turbo Source Search	15
125	V-Communications	25
77	Wendin, Inc.	15
73	Whitewater Group	C2
123	Xenosoft	41

* *Contact these Advertisers directly*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Attention Programmers!

HERE'S THE QUICK, EASY WAY TO MAKE SCREENS, DEMOS AND TERRIFIC PROGRAMS!

DisplayExpress™: A fast, efficient and easy-to-use screen manager.

DisplayExpress™ helps you make efficient screens as easily as you would type a letter. Such screens take little space (less than 2 kbytes in the average) and are displayed instantaneously.

Colors...Character graphics...Editing power...Automatic drawing commands...And a very compact run-time support: 6 kbytes only, yet it includes all you need to load and display screens and menus!

The package also includes the DemoExpress® Viewer which allows you to make demodiskettes and slides quickly and easily. Ideal for your presentations, tutorials and continuously running demos.

C-Display Librarian™: The professional programmers' choice.

C-Display Librarian™ provides complete support for the man-machine interface. Performs BIOS video, screen editing, data entry and validation, keyboard control and bit mapped graphics.

Modular design...Small size...Great speed... Professional documentation...Full source... Compatible with Microsoft and Lattice compilers. Available for IBM Personal System/2 or PC/XT/AT.

30-day money-back guarantee!
No royalties on programs or demos!
Free programming examples with source!
Free shipping within the United States on all prepaid orders!

"If you want to include bit-mapped graphics in your application, then this might be the package for you"
(D. Dramheller and M. Wilson, Computer Language)

"DisplayExpress and C-Display Librarian are very fine products and I am looking forward to incorporating them into my applications development"
(R.W. Hirn, Vice-President, NHDS Inc., Chicago, IL)

"DisplayExpress and C-Display Librarian are a great time saver and should be in the tool-kit of every C language programmer"
(D. Wilson, McLean, VA)

YES!

I want to make screens, demos and terrific programs right now!

Please rush me:

☐ **DisplayExpress**™ **$69**

☐ **C-Display Librarian**™ **$69**

Sales tax (N.Y. residents only) $ _____

TOTAL $ _____

ALL FOREIGN ORDERS: add $10.00 per product and make payment by bank draft payable in US dollars and drawn on a US bank.

NAME _____

COMPANY _____

TEL.: _____

ADDRESS _____

CITY _____

STATE _____ ZIP _____

COUNTRY _____

New! Low Price!

DisplayExpress™ **$69**

C-Display Librarian™ **$69**

SYDETECH
SYSTEM DEVELOPMENT TECHNOLOGIES INC.
43-23 Colden Street, #17C • Flushing, New York 11355

For COD orders or the name of the nearest dealer please call (718) 886-0221

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Borland's new Turbo C: The most powerful optimizing compiler ever

Our new Turbo C® generates fast, tight, production-quality code at compilation speeds of more than 13,000* lines a minute!

It's the full-featured optimizing compiler everyone has been waiting for.

Switching to Turbo C, or starting with Turbo C, you win both ways

If you're already programming in C, switching to Turbo C will make you feel like you're riding a rocket instead of pedaling a bike.

If you've never programmed in C, starting with Turbo C gives you an instant edge. It's easy to learn, easy to use, and the most efficient C compiler at any price.

"Turbo C does look like What We've All Been Waiting For: a full-featured compiler that produces excellent code in an unbelievable hurry . . . moves into a class all its own among full-featured C compilers . . . Turbo C is indeed for the serious developer . . . One heck of a buy—at any price."

Michael Abrash, Programmer's Journal

Join more than 100,000 Turbo C enthusiasts. Get your copy of Turbo C today!

Number **3** on the Reader Service Card

All Borland products are trademarks or registered trademarks of Borland International, Inc., or Borland/Analytica, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.
Copyright 1987 Borland International BI-1136

Technical Specifications

- ☑ *Compiler: One-pass optimizing compiler generating linkable object modules. Included is Borland's high-performance Turbo Linker.™ The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).*
- ☑ *Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.*
- ☑ *Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.*
- ☑ *Links with relocatable object modules created using Borland's Turbo Prolog® into a single program.*
- ☑ *Inline assembly code.*
- ☑ *Loop optimizations.*
- ☑ *Register variables.*
- ☑ *ANSI C compatible.*
- ☑ *Start-up routine source code included.*
- ☑ *Both command line and integrated environment versions included.*
- ☑ *License to the source code for Run-time Library available.*

Sieve benchmark

	Turbo C	Microsoft® C
Compile time	*2.4*	13.51
Compile and link time	*4.1*	18.13
Execution time	*3.95*	5.93
Object code size	*239*	249
Execution size	*5748*	7136
Price	*$99.95*	$450.00

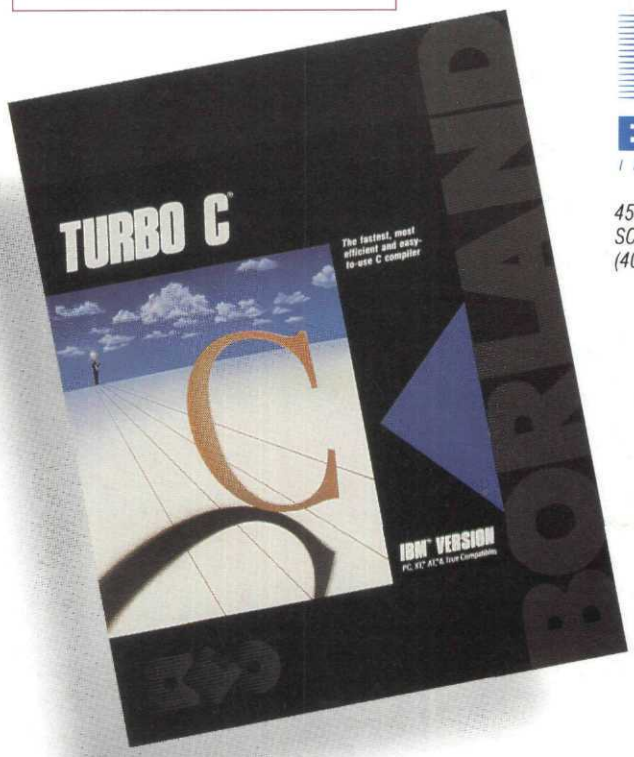
*Benchmark run on an IBM PS/2 Model 60 using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51.

Minimum system requirements: IBM PC, XT, AT, PS/2 and true compatibles. PC-DOS (MS-DOS) 2.0 or later. 384K.

For the dealer nearest you or to order by phone call

(800) 255-8008

in CA (800) 742-1133 in Canada (800) 237-1136



Only $99.95!

BORLAND INTERNATIONAL

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED